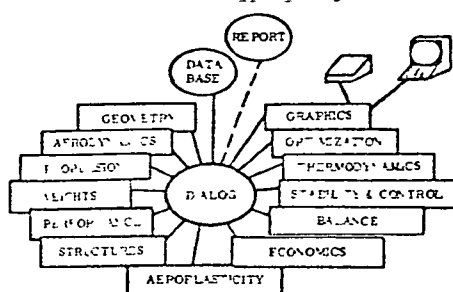


N73. 31143  
CR-134032

AEROPHYSICS RESEARCH CORPORATION  
TECHNICAL NOTE

JTN-01



# CASE FILE COPY

DIALOG:  
AN EXECUTIVE COMPUTER PROGRAM  
FOR LINKING INDEPENDENT PROGRAMS

by C. R. Glatt, D. S. Hague and D. A. Watson

Prepared by  
AEROPHYSICS RESEARCH CORPORATION  
Houston, Texas 77058  
for Johnson Space Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION SEPTEMBER 1973

## PREFACE

This report was prepared under Contract NAS 9-12829, "An Optimal Design Integration of Reusable Launch Vehicles (ODIN/RLV) Computer Program Converted to the Manned Spacecraft Center's UNIVAC 1108 Computer System." The study was carried out in the period from June, 1972, to June, 1973. The study was funded by the National Aeronautics and Space Administration, Johnson Space Center, and sponsored by the Engineering Analysis Division, Flight Performance Section. Mr. Robert Abel served as technical monitor for the study.

## ABSTRACT

A very large scale computer programming procedure called the DIALOG Executive System has been developed for the Univac 1100 series computers. The executive computer program, DIALOG, controls the sequence of execution and data management function for a library of independent computer programs. Communication of common information is accomplished by DIALOG through a dynamically constructed and maintained data base of common information.

The unique feature of the DIALOG executive system is the manner in which computer programs are linked. Each program maintains its individual identity and as such is unaware of its contribution to the large scale program. This feature makes any computer program a candidate for use with the DIALOG executive system. This manuscript describes the installation and use of the DIALOG Executive System at Johnson Space Center. Installation on other Univac series computers would be similar.

## TABLE OF CONTENTS

	Page
1.0 SUMMARY.....	1
2.0 INTRODUCTION.....	3
3.0 DIALOG FUNCTIONS.....	7
3.1 Computer Control Card Assembly.....	7
3.1.1 Execution of an Applications Program.....	7
3.1.2 Creation of a Control Card Data Base (CCDATA).....	10
3.1.3 Execution of a Sequence of Applications Programs through Control Card Linkage.....	12
3.1.4 Repetition of Control Card Sequences.....	13
3.2 Data Management Function.....	18
3.2.1 Data Base Information Transfer System.....	21
3.2.2 Creation of a Design Data Base....	22
3.2.2.1 Adding Information to the Design Data Base.....	24
3.2.2.2 Combining Data Base Information.....	26
3.2.2.3 Defining Variables and Reserving Space in the Data Base.....	29
3.2.2.4 Identification of Applica- tions Program Data.....	29
3.2.3 Communicating Information from the Data Base to the Applications Programs.....	31
3.2.3.1 Modifying Program Input to Communicate with the Data Base.....	31
3.2.3.2 Data Base Communication through Input.....	31
3.2.3.3 Combining Data Base Information in the Modified Input Stream....	33
3.2.4 Communicating Information from the Applications Programs to the Data Base.....	35
4.0 INSTALLATION OF THE DIALOG EXECUTIVE SYSTEM ON A TYPICAL 1100 SERIES COMPUTER.....	37
4.1 Compilation and Storage of the DIALOG Executive Program.....	38
4.1.1 Data Base Parameters.....	39

	Page
4.1.2 Deck Setup for DIALOG Storage.....	41
4.2 Compilation and Storage of a Library of Programs.....	41
4.2.1 Program Modification to Provide Data Base Information.....	43
4.2.1.1 Creating a Special Out- put File.....	43
4.2.1.2 Format of the Special Output File.....	44
4.2.1.3 Use of the NAMELIST Feature in FORTRAN.....	45
4.2.2 Storage of an Absolute Element Program.....	45
4.3 Assembly of the Control Card Data Base...	47
4.3.1 Construction of a Control Card Sequence for a Data Base Entry....	48
4.4 Storage of the DIALOG Executive System...	51
5.0 USE OF THE DIALOG EXECUTIVE SYSTEM.....	54
5.1 Control Directives.....	56
5.1.1 INITIAL Directive.....	59
5.1.2 RESTART Directive.....	59
5.1.3 UPDATE Directive.....	60
5.1.4 DESIGN Directive.....	62
5.1.5 EXECUTE Directive.....	62
5.1.6 LOOP TO Directive.....	63
5.1.7 IF Directive.....	63
5.1.8 PRINT Directive.....	63
5.1.9 END Directive.....	64
5.2 Communication Commands.....	64
5.2.1 The ADD Command.....	65
5.2.1.1 Adding Fixed Element Information.....	67
5.2.1.2 Adding Multiple Data Elements.....	67
5.2.1.3 Transferring Data Elements.....	70
5.2.1.4 Combining Data Elements with Constants.....	70
5.2.1.5 Combining Data Elements with other Data Elements and Constants.....	71
5.2.1.6 Adding Arrays.....	72
5.2.1.7 Adding Constant Arrays...	72
5.2.1.8 Adding Mixed Arrays.....	73
5.2.1.9 Transferring Array Elements.....	73

5.2.1.10	Combining Array Elements.....	74
5.2.2	The DEFINE Command.....	74
5.2.3	The Comment Command.....	75
5.2.4	Replacement Command.....	77
5.2.4.1	Simple Replacement of Data Base Names.....	79
5.2.4.2	Simple Replacement of Data Base Combinations....	83
5.2.4.3	Array Replacement by Name.....	84
5.2.4.4	Array Replacement of Data Base Combinations.....	85
5.3	Standard Utility Procedures.....	85
5.3.1	COMPL1/MYPG1: Compile a FORTRAN Program/Execute the Compiled Program.....	86
5.3.2	REPORT: Generates Data Base Status Report.....	86
5.3.3	ENDODN: To Save a Design Data Base for Future Use.....	89
5.4	Special Options in DIALOG Executive Program.....	89
6.0	APPLICATIONS.....	92
6.1	Orbiter Landing Skin Temperature Study...	92
6.2	Shuttle Orbiter Wing Design Study.....	92
7.0	CONCLUSIONS.....	100
8.0	REFERENCES.....	102
APPENDIX A	CONTROL DIRECTIVE SUMMARY.....	103
APPENDIX B	COMMUNICATION COMMAND SUMMARY.....	104
APPENDIX C	EXCLUDED NAMES FOR DATA BASE VARIABLES.....	106

## 1.0 SUMMARY

An executive computing system called DIALOG has been developed for linking independent applications computer programs to form an interdependent system of programs for synthesizing engineering processes.

All elements of the program intercommunication are directly controlled by DIALOG. The significant advantage to the system is the rapid response to everchanging synthesis requirements. The analyst has the choice of model complexity through replacement or addition of functional program elements. The developer of new program elements is unconstrained by the requirements of the executive system. New programs may be rapidly incorporated into the program library.

The DIALOG executive system represents a significant departure from the usual means of forming synthesis programs as illustrated in Figure 1-1. Today's typical synthesis program is a collection of analysis programs merged together into a single computer program. The data management function is programmed into the synthesis program. DIALOG controls the sequence of execution of the independent program elements and performs the data management function by maintaining a data base of information. The data base is the common information link among the program elements. The basic elements of the DIALOG executive system are:

1. A library of independent applications programs.
2. A data base of control card sequences for the execution of the independent programs.
3. A language for controlling the execution of a sequence of independent programs by simple commands.
4. A dynamically constructed data base containing all interprogram data in an unstructured name oriented format. These data can be randomly selected by name at any point in the simulation.
5. A language for automatically retrieving data base information as input to any of the applications programs. An advanced information access and retrieval system is included as an integral part of the DIALOG executive system.
6. A simple technique for allowing any program in the synthesis to update the data base. The technique does not influence the stand alone operation of the program.

INDEPENDENT  
APPLICATIONS  
PROGRAMS

TYPICAL  
SYNTHESIS  
PROGRAM

DIALOG  
EXECUTIVE  
SYSTEM

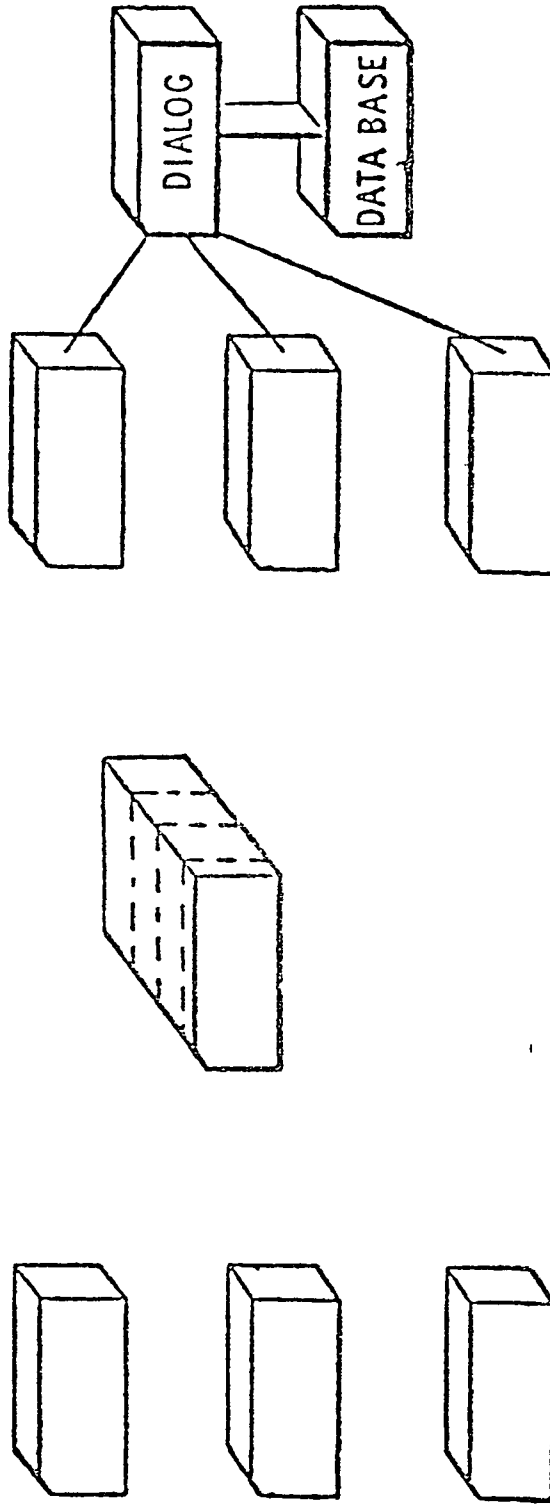


FIGURE 1-1 LARGE SCALE PROGRAMMING CONCEPTS

## 2.0 INTRODUCTION

The design of an aerospace vehicle demands the involvement of specialists from all engineering disciplines. Many iterations are usually required before a suitable vehicle design emerges. The design iterations usually require from one to three months depending on the level of detail employed. Each discipline involved in the design process generally is constrained by the requirements of other disciplines, and much laborious data communication is required at each step. The interface among disciplines is often ill-defined leading to untimely or inaccurate information transfer. Under these circumstances, decisions affecting the usefulness of the end product can be based on poor or unreliable information.

The above factors have lead to increased use of the high speed digital computer to expedite the design process and improve the quality of the design information. Automation of the individual disciplines has played an increasing role in the design process for more than a decade. Structural analysis and system performance have led the way in large scale computer applications, although nearly every aspect of the design process has been automated to some degree. More recently, the merging of the technologies into a single preliminary design tool has been attempted. One successful preliminary design tool is exemplified by References 1 and 2. Here a complete synthesis of the design and mission analysis is contained in a single computer program.

The confidence gained in early simulation attempts has led to the development of more detailed and complex modules. References 3 through 8 are examples of recently developed simulation tools. However, most modern day integrated design programs tend to suffer from one or more of the following deficiencies:

- a. Lack of depth in the analysis techniques.
- b. Insufficient or inflexible data intercommunication.
- c. Poor response time to rapidly changing design requirements.
- d. Excessive computer core requirements.

By-and-large the technical depth is available in independent technology programs. The pattern of development of these programs has been the generalized multiple option approach suitable to the analysis of many classes of vehicles, each

class being represented by input data. The problem arises in combining the technology programs into a design synthesis program. Computer core limitations require that resulting synthesis programs be generally more limited in scope than the individual program. As a result, the synthesis programs tend to become obsolete very quickly as the design process evolves. Very often the obsolescence occurs before any effective use can be made of the synthesis program.

The deficiencies described above have led to the development of a new design synthesis procedure called ODIN (Optimal Design Integration) described in Reference 9. The ODIN procedure shown schemetically in Figure 2-1 is a very large scale synthesis procedure which allows the selective use of existing computer programs as elements of a larger more comprehensive design simulation. Reference 9 exemplifies the technology modules which have been used with the ODIN procedure. All the depth of analysis in each technological area is maintained and the computer core requirement is no larger than the largest program element selected.

The linking of the independent program elements is controlled by the executive computer program, DIALOG which also controls the communication of information among the independent program elements. An input language to the DIALOG executive system provides the user with the ability to formulate the design problem at the task level in much the same manner as is currently employed in the design process. As much or as little of the design process may be automated as suites the particular application. The design staff directly controls the specific information being communicated from program-to-program and from the design simulation to the design staff.

Since the system uses existing checked out computer codes as building blocks in performing the design tasks, no program development is usually required. The program elements are usually in common use throughout the design staff and therefore readily usable in the design simulation. No more effort is required to establish an automated design sequence than that required to establish a single design cycle by ordinary means. The same computer codes are generally used in either case. Once established the automated procedure can be used many times for design perturbations, and can be quickly changed to suit changing design requirements. The designer never relinquishes his option to perform any task by some alternate means including hand calculation.

The current documentation describes the control and communication language of the DIALOG executive system. It will become apparent that the DIALOG executive system is not

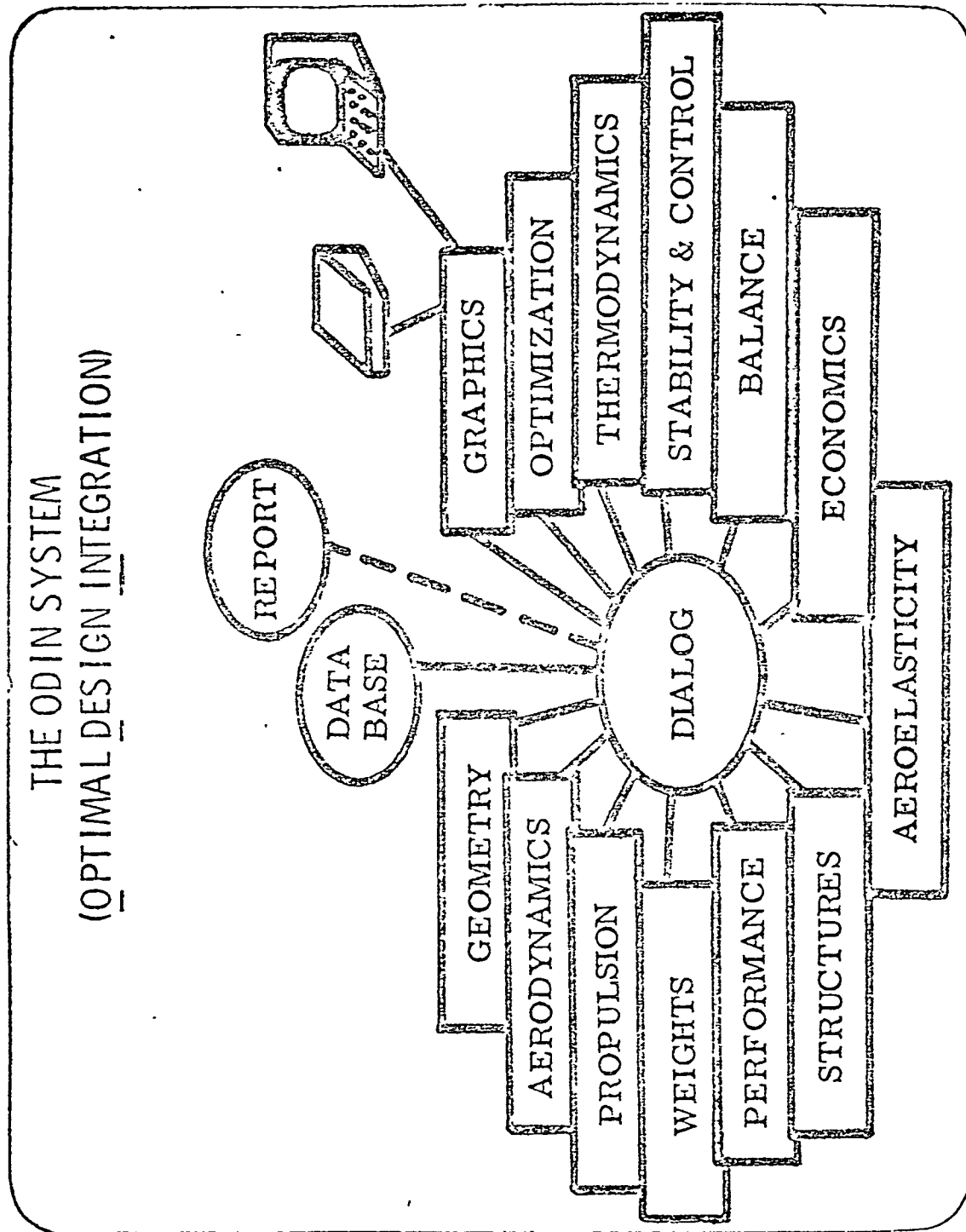


FIGURE 2-1 CONTRIBUTION OF DIALOG TO THE ODIN PROCEDURE

specifically tied to design simulation. Although originally developed to implement the ODIN procedure, the DIALOG executive system is generally applicable to any engineering process. Little reference is made to design simulation in this report. The library programs which DIALOG controls are referred to as applications programs to avoid the implication that only technology oriented tasks may be employed.

Indeed there are many "utility programs" used in ODIN which perform non-engineering tasks but are quite useful in any process.

Section 3 contains a general description of the DIALOG executive functions. Much of the programming detail has been omitted providing an overview of the system capability. Section 4 is a detailed description of the installation of the DIALOG executive system at the NASA Johnson Space Center Computer Complex. The nature of the DIALOG executive system requires a close relationship with the operating system on which it is installed. Although representative of the installation on other Univac systems, the JSC system installation is presented as an example. Section 5 is devoted to the use of the DIALOG languages developed for the purpose of linking independent programs and communicating information among them.

### 3.0 DIALOG FUNCTIONS

Usually the submission of a computational sequence to the digital computer involves the execution of a single computer program with possible repetitive evaluation of successive data cases. When using the DIALOG executive system, submission of a computation may involve the sequential execution of many programs to obtain a complete analysis. For example, the repetitive execution of program sequences will be required for parametric studies or optimization problems. The use of the DIALOG executive system also affords the analyst the opportunity to conveniently communicate data from stratified sources among the programs in the execution sequence. A discussion of these two basic functions is presented in the following paragraphs.

#### 3.1 Computer Control Card Assembly

On a digital computer the execution of a single program is governed by a set of control cards which provides instructions to the computer system for compiling and/or loading the specified program. The control cards are peculiar to each computer system and installation. The control cards rarely employ user oriented format. For example, figure 3-1 presents typical control cards for an elementary FORTRAN compilation and execution of the same program on a Univac 1110, CDC 6000 series computer, and an IBM 360 series computer. Further, control cards on any computer of a given series or manufacturer can vary from installation to installation. Figure 3-2 shows the control cards to retrieve from storage and execute a machine language program at two different installations. Though each installation uses an 1100 series computer, the differences in compilers, loaders and peripheral hardware result in entirely different control cards to perform the same function.

3.1.1 Execution of an Applications Program. - In actuality, to retrieve and execute an application program, several independent programs must be executed. Collectively the control cards required to execute an applications program may be referred to by name such as PGMA or PGMB. These independent program executions which we call "control cards" are all part of the computer operating system. System programs of the type called

### CDC 6000 SERIES COMPUTING SYSTEM

```
RFL,60000,  
FTN,OPT=0,  
LGO,  
7-8-9  
        SOURCE DECK  
7-8-9  
        DATA DECK  
6-7-8-9
```

### IBM 360/67 SERIES COMPUTING SYSTEM

```
//EXEC FORTGCG  
//FORT.SYSIN DD *  
        SOURCE DECK  
/*  
//GO.SYSIN DD *  
        DATA DECK  
/*
```

### UNIVAC 1108 SERIES COMPUTING SYSTEM

```
@ FR5 MAIN  
        SOURCE DECK  
@ XQT MAIN  
        DATA DECK  
@FIN
```

FIGURE 3-1 TYPICAL CONTROL CARDS TO COMPILE AND EXECUTE  
A FORTRAN PROGRAM

UNIVAC 1108 EXEC II

@ ASG A = 12012 (PCF TAPE)

@ XQT CUR

IN A

TRI A

@ XQT PGMA

{ DATA DECK }

@ FIN

UNIVAC 1110 EXEC VIII

@ASG,A PGMA (PROGRAM FILE)

@XQT PGMA

{ DATA DECK }

@FIN

FIGURE 3-2 EXAMPLE CONTROL CARDS TO RETRIEVE AND EXECUTE A MACHINE LANGUAGE PROGRAM AT TWO UNIVAC 1100 SERIES COMPUTER INSTALLATIONS.

by control cards bear a similar relationship to the computer operating system as do independent applications programs PGMA and PGMB to the DIALOG executive system, figure 3-3. This analogy may be formalized as follows:

"The operating system employs independent system utility programs to retrieve, compile and execute a given applications program. The DIALOG executive system employs control card sets to synthesize an engineering process."

DIALOG contains a higher order programming language which carries out the analysis function by linking control card sets rather than carrying out the individual control card functions.

3.1.2 Creation of a Control Card Data Base (CCDATA). - The nature of the computer operating systems with regard to the execution of applications programs via a sequence of control cards has led to the development of the control card data base concept. The data base contains all the control card sequences required to retrieve and execute a library of applications programs. Collectively, any sequence of control cards necessary to execute a given applications program is referred to by a name.

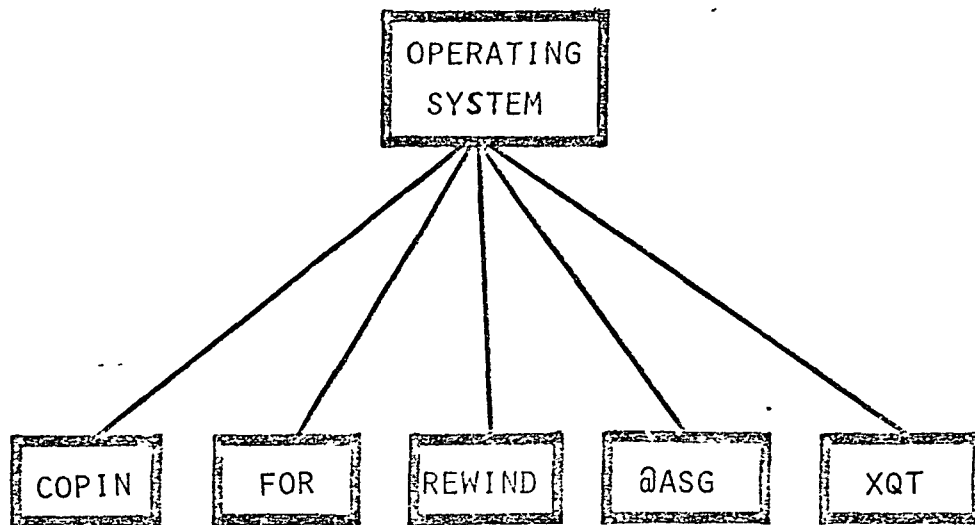
EXECUTE PGMA

The name, PGMA, is assigned when the control card sequence is stored in CCDATA. In the remainder of this section details of the control cards will be omitted. The control card sequences will be referred to by the name under which it is stored. The command to execute the control card sequence:

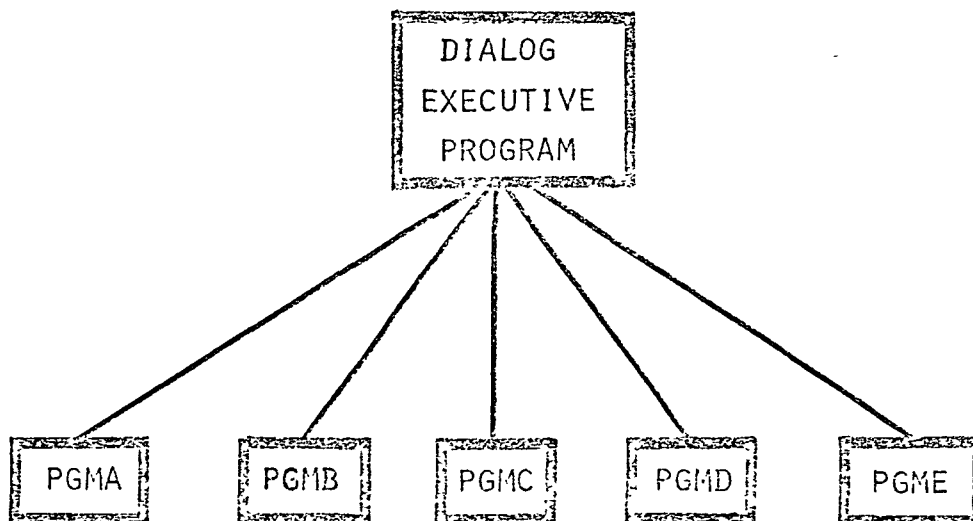
EXECUTE PGMA

will be referred to as a control directive (i.e. the EXECUTE directive). Other control directives will be described, each performing a "control function" in the execution sequence. Collectively the control directives form the DIALOG control directive language.

The control directive language is input to the DIALOG executive program. DIALOG processes all input to all programs and performs certain functions based upon the control directives encountered. To distinguish DIALOG



COMPUTER OPERATING SYSTEM



DIALOG EXECUTIVE SYSTEM

FIGURE 3-3 ANALOGY BETWEEN OPERATING SYSTEM AND DIALOG EXECUTIVE SYSTEM

input from the input of applications programs, the control directives must be delimited as follows:

```
'EXECUTE PGMA'
```

The (') is used here to represent a 2-8 punch.

The creation of the control card data base is a function of DIALOG. It reads from input cards the control card sequences to retrieve and execute programs from the library. The control directive which creates the control card data base is:

```
'CREATE CCDATA'
```

Following this directive, the control card sequence such as those illustrated in figure 3-1 and 3-2 are entered into CCDATA by name:

```
'PGMA =  
      control  
      card  
      sequence
```

Any number of these control card sequences may be entered into CCDATA, each representing the retrieval and execution of an applications program. Once established, the control card data base can be freely accessed by the EXECUTE control directive without regard to the actual control card sequence involved. Usually the control card sequences do not change. However, the DIALOG control directive language permits the complete replacement of existing control cards sequences or the modification of individual cards. The UPDATE control directive is used for this purpose. UPDATE is described in detail in Section 5.

3.1.3 Execution of a Sequence of Applications Programs through Control Card Linkage. - Now consider the problem of sequential execution of more than one application program using the DIALOG execution system. Assume the following three control directives:

```
'EXECUTE PGMA '  
'EXECUTE PGMB '  
'EXECUTE PGMC '
```

The function of DIALOG is simply to retrieve the control cards sequences for PGMA, PGMB and PGMC from the control card data base, CCDATA and queue them sequentially on a file called CONTROL. The CONTROL file is then interrogated by the operating system which performs the various control card functions. Included in these control card functions are the executions of the desired applications programs. Figure 3-4 illustrates the relationship between the DIALOG executive and the operating system. Progressing from left to right in figure 3-4, the control directives to execute control card sequences are read by DIALOG, which "builds" the control card sequences from information stored in the control card data base and passes the control card sequences to the operating system.

The physical link among the control card sequences of the various applications programs and DIALOG is an operating system utility called the ADD utility on the Univac 1110. ADD is executed by a control card which forces the operating system to read control cards from an alternate file built by DIALOG. On some system the ADD file is physically merged with the input stream. In the DIALOG executive system ADD is used to link the execution sequences of an applications program to the execution sequence for DIALOG, then from DIALOG to the next applications program, then to DIALOG, etc. This is illustrated in Figure 3-5. So the DIALOG executive program first constructs the control card sequences to execute applications programs then through the use of ADD, provides for the re-execution of DIALOG to process the application program input and output data. Details of the operation of the utility ADD are contained in the Univac 1108 operating systems manual.

3.1.4 Repetition of Control Card Sequences. - Thus far we have described a capability which permits the execution of control card sequences in an arbitrary manner using higher order readily understood commands. This is achieved by the creation of a control directive language which replaces the control card sequence such as those in figures 3-1 and 3-2. However, there are two additional capabilities which exist in the DIALOG executive system which are not possible simply by queuing control card sequences. These include the conditional branching logic described below and the maintenance of a design data base described in Section 3.2.

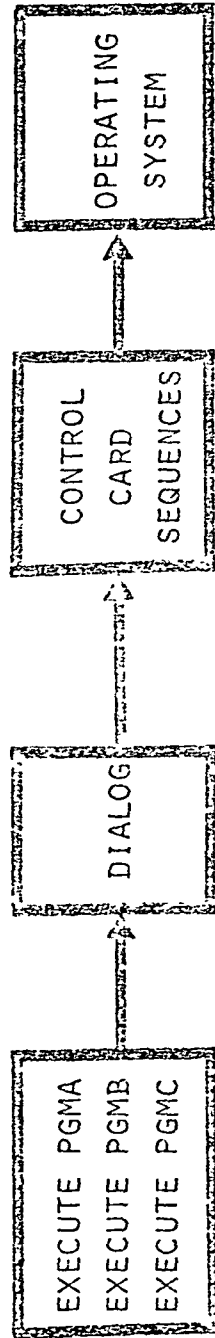


FIGURE 3-4 ILLUSTRATION OF THE RELATIONSHIP BETWEEN THE DIALOG EXECUTIVE  
AND THE OPERATING SYSTEM

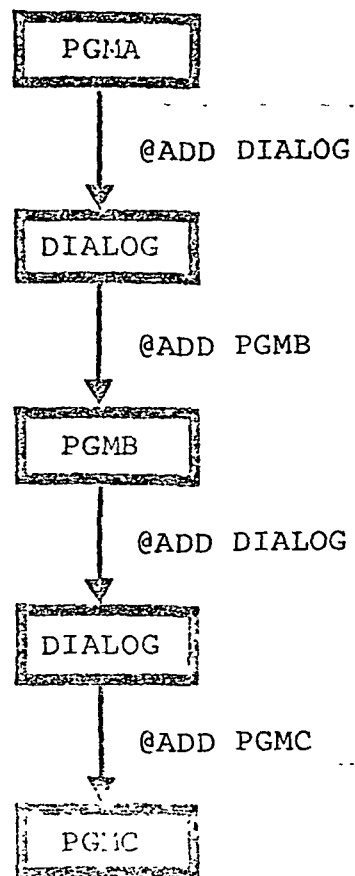


FIGURE 3-5 ILLUSTRATION OF THE FUNCTION OF THE SYSTEM  
UTILITY, ADD

The DIALOG executive system permits automatic repetition of control card sequences by a system of conditional branching logic. This capability is achieved by extension of the control card directive language in the following manner:

'DESIGN POINT1'

'LOOP TO POINT1'

'IF V1 .LT. V2'

The DESIGN directive establishes an identifier in the execution sequence where control may be returned (or skipped to). The LOOP TO directive points to the identifier to which control is to be returned. The IF directive is a conditional operator based upon design dependent logic. If absent, the LOOP TO directive is a mandatory branching command. V1 and V2 are example values which may be constant or computed in any of the applications programs. In the latter case such values must have variable names and be defined in the design data base. A description of the design data base is given in Section 3.2.

In general, the DIALOG executive system permits a complicated system of analysis loops for satisfying a variety of matching constraints. It is not possible or necessarily desirable to rigidly define the topology of the system of computational loops. Instead, the analysis sequence to be performed is defined within the control directive language. This technique allows the analyst complete freedom in specifying the computational sequence; no limit is placed on the complexity of the analysis.

Any number of loops can be created using the LOOP TO and conditional IF control directives and the associated DESIGN control directive. The IF tests employed encompass the standard set of six tests in FORTRAN; although the form of the DIALOG control directive language test differs in form to that of FORTRAN. The six tests are:

'IF V1 .LT. V2'	<u>IF (V1 &lt; V2)</u>
'IF V1 .GT. V2'	<u>IF (V1 &gt; V2)</u>
'IF V1 .LE. V2'	<u>IF (V1 ≤ V2)</u>
'IF V1 .GE. V2'	<u>IF (V1 ≥ V2)</u>
'IF V1 .EQ. V2'	<u>IF (V1 = V2)</u>
'IF V1 .NE. V2'	<u>IF (V1 ≠ V2)</u>

As noted previously V1 and V2 are constants or variables constructed in the design data base or constructed within any independent program in the synthesis and passed to the design data base.

The ability to select alternative program execution paths based on design dependent logic is illustrated below:

```

.....
'EXECUTE PGMA'
'DESIGN POINTA'
'EXECUTE PGMB'
'LOOP TO POINTB'
'IF V1 .EQ. V2'
'IF V3 .LT. V4'
'EXECUTE PGM C'
'LOOP TO POINTA'
'DESIGN POINTB'
'EXECUTE PGMD'

```

The above control directive sequence defines the execution of PGMA and PGMB with a conditional loop (in this case, skip) to POINTB. If neither of the IF conditions

is satisfied, PGM C is executed followed by a mandatory loop back to POINT A.

In general, both V1 and V2 may be defined by the analyst or alternately either may be a variable computed by any of the application programs. In the latter case such variables must be defined in the data base as described in Section 3.2.

### 3.2 Data Management Function

The usual manner of transferring information from one program to another is by use of a structured file. This simply means program A is coded to create a file of data in exactly the same format as required by program B. In the sequential execution of the two programs, the structured file created by A is passed to B via control card procedures.

In the DIALOG executive system the above means of transfer of information is possible and often employed. However, in the communications of information from one program to another, it is not always possible or even desirable to create a structured file of input data for each program. Often only a few stratified bits of common information are required for each program. Usually a different set of data and a different order is required for each program. The DIALOG executive system maintains a name-oriented data base containing an unstructured set of data accessible by all programs.

The data base file of information can be dynamically constructed and altered by DIALOG as the analysis proceeds. Construction of the design data base involves the following tasks:

1. Search to see if the variable name exists in the data base.
2. If not, locate a vacant location in the data base and install the information and the name.
3. Otherwise, replace the information associated with the name.

Additional information can be added or existing information may be updated by the analyst or by an applications program at any point in the analysis (data base limits are discussed in Section 4). Updating the data base by

the applications programs involves tasks similar to those described above. Figure 3-6 illustrates the data interplay among the applications programs. Consider a variable stored by the name, WEXPAR. Assume WEXPAR is computed in program A (PGMA) and subsequently used by program B (PGMB). Schematically this is illustrated in figure 3-6. Any number of subsequent programs may access WEXPAR or alternately update the value of this variable. All interface between the applications programs and the design data base are performed by the DIALOG executive as illustrated in figure 3-7. The same is true of the interface between the analyst and the data base. All data requests are addressed to the DIALOG executive.

Data base information may be accessed by the analyst for placement into the input stream of any of the applications programs. The DIALOG tasks in performing this function are:

1. Search to see if the variable name encountered exists in the data base.
2. If not, ignore the access request.
3. Otherwise, retrieve the information associated with the name.
4. Replace the variable name encountered with the data base information.

The technique employed in the storage and retrieval of data base information is discussed in the following paragraphs. They involve the extension of the control card directive language described in Section 3.1 as well as the creation of a new intercommunication language for passing design information from one applications program to another.

The new language contains a simple set of instructions which will be referred to as communication commands. Communication commands are physically inserted into the applications programs input data. In general, these commands are either removed entirely or replaced with data by the DIALOG executive program before the input stream is processed by the applications program.



The communication commands form the basis by which unstructured information is passed from one applications program to another. The communication commands are delimited in exactly the same manner as the control directives. Complete syntax rules for the language are given in Section 4.

3.2.1 Data Base Information Transfer System. - Data base information transfer is accomplished through a rapid search by name. Search speed is obtained by the use of "hash" and "collision" methods of reference 10. This approach is more efficient than the more usual linear sequential search which starts with the first name in the table and proceeds sequentially until the desired name is located and the corresponding value is retrieved.

The hash and collision data transfer system operates in the following idealized manner:

1. Take the variable name and treat the binary representation of this word as an integer.
2. Find the remainder when the integer representation is divided by the number of elements in the data base. This is equivalent to the FORTRAN MOD function which is a very rapid machine operation.
3. Use the remainder as the nominal location or "hash" location of the variable within the data base. This assures the location derived will fall within the data base limits.
4. Check to see if the location is in use since more than one variable name may hash to this location. If this location has already been used for another variable name store the new variable in the next vacant location and provide a pointer to this location in the data base entry originally searched. This pointer is called a "collision" pointer. Each entry has associated with it a name, a value, a hash address and a collision pointer.
5. The retrieval process operates in the same manner. The name is converted to nominal retrieval location. If that location contains the wrong name, the specified alternate location is searched for

the desired name, etc. until the desired name is found and the variable value is retrieved.

Figure 3-8 illustrates the hash and collision method. Suppose the binary representation of three variables A, B and C are identical to a fourth variable stored in the data base. Upon initial entry, the name A "hashes" to the occupied location. After unsuccessful comparison with the existing name at that entry, a new location for A is defined and a collision pointer is stored at the original entry forming a link to the new location. Once a location is established for A, the information (value) is stored or retrieved. The name B is also "hashed" to the original location. An unsuccessful comparison with the existing entry sends B to the location where A is stored via the pointer described above. An unsuccessful comparison with A causes the next available location to be defined for B. A pointer to the newly defined location is stored at the entry for A forming the link to B. This chaining process described can be continued to the limits of the table.

In numerical experiments with a 2000 word data base filled approximately 75 per cent, it was found that the average name can be retrieved in less than two attempts (fetches). This would compare with 750 fetches using a linear search for information retrieval. In practice using DIALOG, approximately 9000 values per second may be retrieved on the CDC 6600. This figure varies but is less affected by the amount of data stored than by the internal numerical pattern produced by the variable names stored. It is difficult to control numerical uniqueness since the data base names are arbitrarily chosen by the user. However, in all past experiences with DIALOG collisions have never exceeded 20 per cent. The access time is essentially unchanged with data base size, while the linear search technique increases in access time in proportion to data base size.

3.2.2 Creation of a Design Data Base. - The design data base, DBASE, is created in much the same manner as the control card data base. The two data bases are similar in construction and occupy the same computer core locations but at different times. The data bases consist of two distinct parts, a free storage array of packed information and a directory of names and pointers to the actual data in the free storage array. Space in the free storage array is allocated as required by the user and/or the applications

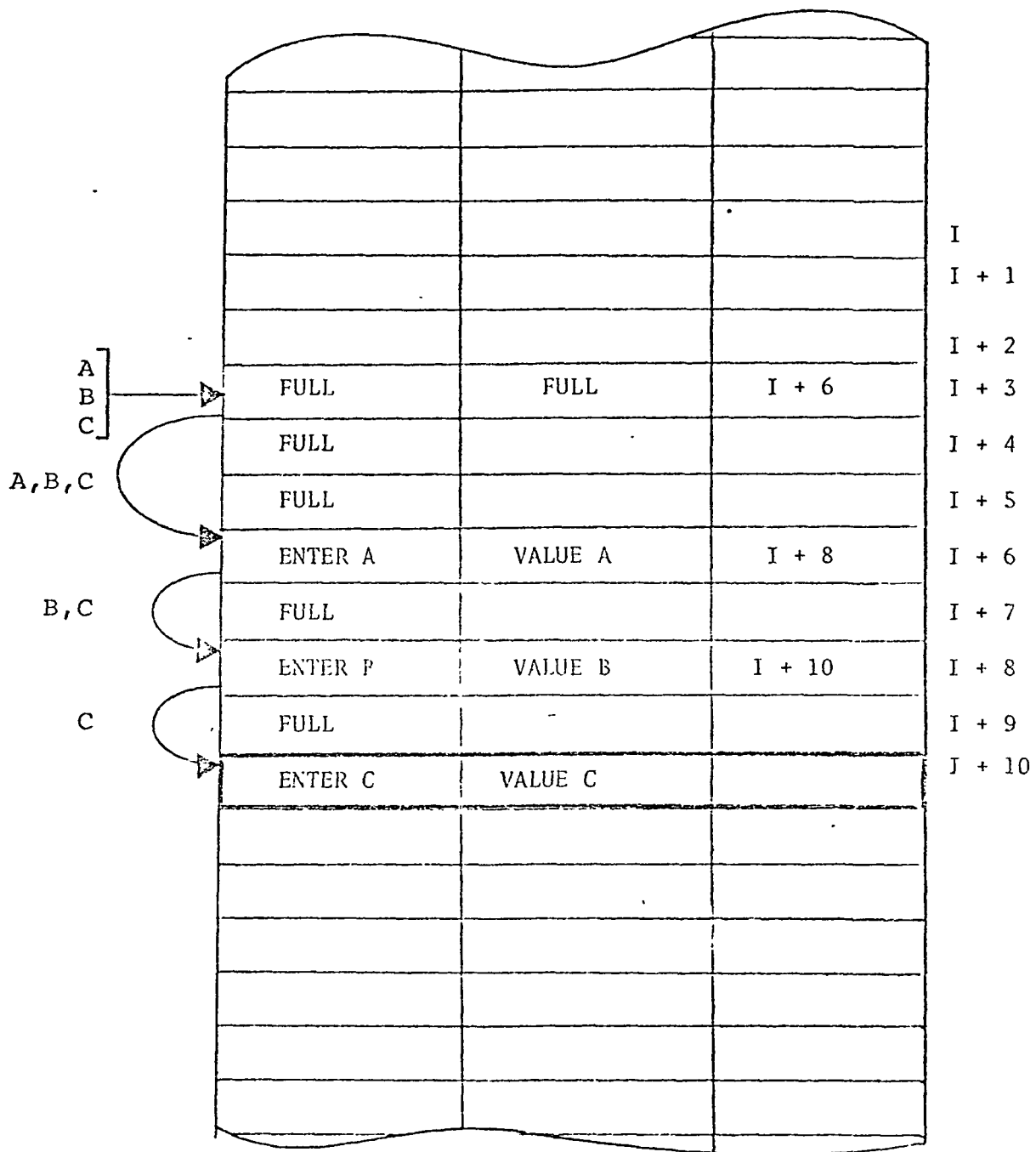


FIGURE 3-8 IDEALIZED INFORMATION RETRIEVAL SYSTEM

programs. As the information is stored in the free storage array, the directory is constructed as described in Section 3.2.1. Access to the data base is always through the directory. Both the directory and the data base have certain attributes which distinguish them from one another. Among these are:

- Total number of data base entries.
- Number of computer words per data base entry.
- Total number of directory entries.
- Number of words of descriptive information associated with each name.

For example, CCDATA elements are 8 computer words (8 words = one card) in length since CCDATA is used for storing control card sequences. DBASE elements are two computer words in length since DBASE is used for storing design type data in BCD format.

The construction techniques employed in creation of the design data base as well as the control card data base are easily extendable to a multiple data base involving many combinations of attributes such as data type and technology origin.

The design data base, DBASE, is created with the control directive:

```
'INITAL DBASE'
```

This directive is followed by a file of information containing the necessary communication commands to initially establish the data base. These communication commands are described below. It is not essential that any information be initially placed in the data base. The dynamic nature of data base maintenance permits information to be added at any point in the execution sequence.

3.2.2.1 Adding Information to the Design Data Base. - The basic communication command available to the analyst is the ADD command; not to be confused with the ADD utility program discussed earlier. It permits a variable name and value or values to be placed in the data base:

```
'ADD name = value, value,'
```

Any number of values may be added for a given name. The number of values associated with the name is also the number of locations reserved in the data base for that information. Later modifications to the information can not create more data base space. The values may be real, integer, hollerith or logical. The data type is immaterial since the information is stored in coded or character format.

A single ADD command may be used for creating or updating many information sets.

```
'ADD V1 = 25., V2 = 30, V3 = ALPHA, V4 = .TRUE.,  
      A = 10., 15., 20., 25., I = 4, 5, 6'
```

The data type is specified by the input. Any of the four common types of variables may be entered into the data base. The format of the ADD statement is patterned after the FORTRAN NAMELIST feature and indeed has the same characteristics and utilization rules. For example, all name/value sets are separated by commas (,); all elemental values of an array are separated by commas; the entire statement (command) is delimited. In the case of NAMELIST, the delimiter is a dollar (\$) sign; in the case of ADD command the delimiter is (').

However, the ADD command has additional capability not present in the FORTRAN NAMELIST feature. The value associated with the ADD name may be a previously defined data base variable name:

```
ADD V1 = V2,
```

The affect of the above command is to transfer the information associated with V2 to the data base space assigned to V1. V1 may or may not exist prior to the ADD command. If V1 did not exist, space will be created in the data base as the information is transferred. If V1 did exist, then the information in V1 will be replaced by the information in V2. The transfer of information from one data base location to another is generally limited to scalar quantities. Complete rules are given in Section 5. Finally, the ADD command may be used for transferring multiple constants in the data base:

```
'ADD V1 = 5 * 0.,'
```

In the above illustration, V1 will be a data base array name. Five zero values will be stored.

3.2.2.2 Combining Data Base Information. - The ADD command capability thus far described includes the addition or modification of data base information with either constant or variable type information. The ADD command may also be used for combining existing data base information with other data information or constant information:

'ADD V1 = V2 \* K,'

or

'ADD V1 - V2 \* V3,'

In the above statements V1, V2 and V3 are data base variables and K is a constant. V1 may be a new or existing data base variable. The operation illustrated above indicates a multiplication of the two numbers on the right side of the equal (=) sign prior to transferring the resulting information to the space allocated to V1. Any algebraic operator may be employed as follows:

- + addition
- subtraction
- \* multiplication
- / division
- \*\* exponentiation

More than one operation may be performed on the right side of the equal (=) sign.

'ADD V1 - V2 + V3 \* K,'

Up to ten operations may be performed within a single ADD command. However, the hierarchy or order of the operations is not the same as FORTRAN. For example, in the above illustration, V3 is added to V2, then the sum is multiplied by K. This is a significant departure from the hierarchy employed in FORTRAN. The basic rule in combining variables with the ADD command is:

"The operations are performed in a serial manner analogous to that employed in a hand calculator."

The operations start from the equal sign and progress to the right. The first variable is combined with the second. The result of that operation is combined with the third.

The result of that operation is combined with the fourth, etc. It is very important from the outset that the analyst understand this principle.

In summary, the ADD command gives the analyst the ability to add, modify and combine information in the data base at any point in the execution sequence. The only constraint is that its occurrence must be within an input data set for an applications program or as a result of a CREATE or UPDATE control directive for DBASE. Figure 3-9 illustrates the possible locations for ADD commands. ADD commands can not be mixed with control directives. It may be noted that the ADD command serves as an instruction only to the DIALOG executive and is not a part of the normal input data to the applications program. As such, the ADD command is removed from the input stream as it is processed.

The combining of variables is very useful when coupling computer programs from independent sources. One example is the matter of units conversion. Very often computer programs use different unit systems. When coupling such programs the output of one computer program may not provide compatible data for the input to another. The ADD command gives the analyst an immediate means of providing that essential data compatibility without modifying any applications programs. The DIALOG executive system does not preclude the possibility of automatically providing data compatibility among applications programs at a future date.

The ADD command is also useful for performing simple interface transformations. Indeed, a limited FORTRAN capability exists as part of the communication command language. However, it is not intended to replace FORTRAN or other languages. It simply augments existing analysis tools. Later discussions will show that full FORTRAN (or any other common language) capability is immediately available within the DIALOG executive system for complex data transformation problems.

(A) ADDING DATA TO A NEW DATA BASE  
'INITAL DBASE'

.....  
.....

'ADD V1 = 25., V2 = 30.,'

.....  
.....  
.....

'VN = 60.,'

.....  
\$EF\$

(B) ADDING DATA TO AN EXISTING DATA BASE  
'UPDATE DBASE'

.....  
.....  
.....

'ADD V1 = 25.,'

.....  
\$EF\$

(C) ADDING DATA DURING THE EXECUTION SEQUENCE  
'EXECUTE PGMA'

.....  
.....

'ADD V1 = V2.,'

.....  
\$EF\$

FIGURE 3-9 POSSIBLE LOCATIONS FOR THE ADD COMMAND

3.2.2.3 Defining Variables and Reserving Space in the Data Base. - It is often desirable in using the DIALOG executive system to reserve space in the data base before the information is actually generated. The DEFINE command was developed for this purpose. As with the ADD command, the DEFINE command may be employed anywhere within the execution sequence. However, it is most likely to be used in conjunction with creating or updating the data base. The format is as follows:

```
'DEFINE V1 = n, description,'
```

V1 is a new or existing data base entry. The number of locations reserved is n. If V1 is an existing variable, n is ignored. If n is absent from the command, one is assumed. The description is a short hollerith description briefly describing the variable V1. The length of the description is typically three computer words. This is not a hard limit and can be altered with the alteration of a dimension statement in DIALOG. The description is printed, together with the data name and value when the control directive:

```
'PRINT DBASE'
```

is employed. Figure 3-10 is an illustration of a printed data base.

3.2.2.4 Identification of Applications Program Data. - In addition to the ADD and DEFINE commands, there exists a special "command" for identifying data. The format is:

```
'. comment'
```

No action is performed as a result of this command. It is useful only as an identifier for other data. For example, consider an application program which uses formatted input (i.e. numbers with no identifiers or names associated with them). The comment command may be used to identify the data elements within the input data stream. The affect of processing this command through DIALOG is that the command is simply replaced with blanks. If the resulting card is entirely blank, then the card is "removed" from the input stream.

The affect of using the comment card with the execution sequence is to provide some self documentation. It serves to identify data which is not generally recognizable as it

***** THERE ARE CURRENTLY 19 DATA BASE ENTRIES AS FOLLOWS *****					
NAME	LOCATION	DIMENSION	CURRENT VALUE(S)	ORIGIN	DESCRIPTION
AESOP	39	1		DEFINE	AESOP USED IN SIMULATION
ALP-A	7	2	0.525E+06	AESOUT	AESOP CONTROL PARAMETERS
			0.2715E+01		
BUILD	19	1	0	OFFINE	DYNAMIC D <sub>0</sub> BUILD OPTION
CARDOP	1	1	32	ADD	CARD SKIP OPTION
CAUSEOP	31	1		OFFINE	CAUSES DATA BASE DUMP
DISPLAY	13	1	WPAYLO	INITIAL	SCOPE DISPLAY FUNCTION
ELAPSED	23	1	25.0920000	OFFINE	TOTAL ELAPSED TIME
EVALUAT	25	1	1	AESOUT	BEST PERFORMANCE EVALUATION
EVALU	17	1	0	AESOUT	AESOP EVALUATION COURSEP
EVALU	3	1	2.71499999999999999200	ADD	PROPELLANT MIXTURE RATIO
EVALU	21	1	000IN JAN 22, 72	OFFINE	IDENTIFICATION FOR DIALOG RUN
EVALU	32	1	48673.95593780665652	ADD	TOTAL COST OF BOOSTER
EVALU	27	1	0.19237115923677E+05	DATAOUT	TOTAL COST OF ORBITER
EVALU	5	1	525000.00000000000000	ADD	ENGINE VACUUM THRUST
EVALU	29	1	0.91561786514272E+06	WTOUIT	BOOSTER STAGING VELOCITY
EVALU	33	1	0.250500077074E+05	WTOUIT	ORBITER STAGING VELOCITY
EVALU	11	1	0.26662821532874E+07	WTOUIT	GROSS WEIGHT OF BOOSTER
EVALU	15	1	0.71802039902725E+06	WTOUIT	GROSS WEIGHT OF ORBITER
EVALU	27	1	0.3184776197751E+05	WTOUIT	PAYLOAD OF THE LAUNCH SYSTEM
*****					
Data base entry name	Data base location and dimension of information	Current data base information		Description entered with a 'DEFINE' command	
The command which last updated the information the information in the data base. ODIN output namelist names are treated as commands.					

FIGURE 3-10 EXAMPLE OF PRINTED DATA BASE FOR 19 ENTRIES

stands. The comment command is somewhat analogous to the comment card in the FORTRAN language.

3.2.3 Communicating Information from the Data Base to the Applications Programs. - The three commands described above, the ADD command, the DEFINE command and the '.' command (comment), basically provide user interface with the data base. This paragraph and the next paragraph deal with applications programs interface with the data base.

In this section the passing of information from the data base to the applications programs will be discussed. Section 3.2.4 deals with the passing of information from the applications programs to the data base.

3.2.3.1 Modifying Program Input to Communicate with the Data Base. - Development of the DIALOG executive system is based on the premise that independent applications programs can be made to communicate with each other without significant modification through a data base. By following this premise a method of communicating data base information into each program has been devised. No modification to the program input data code is required by this method. The input data prepared by the user, however, is modified to indicate data base inputs. The modified data input does not affect the applications program since the DIALOG executive program inspects the data input prior to execution of the applications program. DIALOG combines the required data base information with the basic program inputs, then prepares automatically a file containing the modified input format for the applications program and provides for the execution of that program in the nominal manner. This is illustrated schematically in figure 3-11.

It should be noted that the applications program may still be executed in the normal manner as a stand-alone program independent of the DIALOG executive system.

3.2.3.2 Data Base Communication through Input. - Data base information is entered into the applications program input by means of the special delimiters (''). Any data base variable name may be entered between the delimiters. The DIALOG executive program will replace the variable name by its data base value and rewrite a normal card image to replace the modified input cards. The value is placed within the closed region which includes the delimiters.

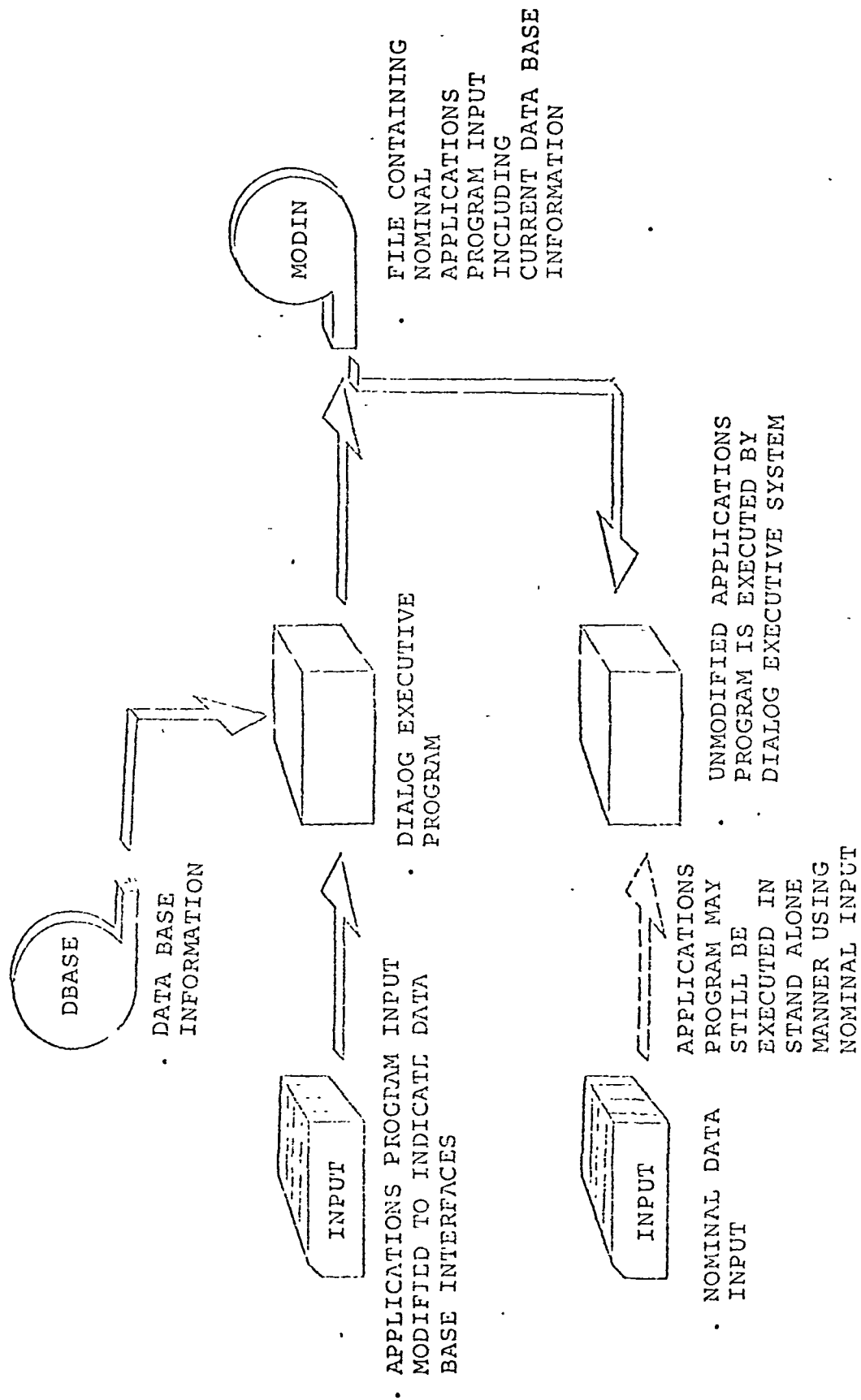


FIGURE 3-11 METHOD FOR MODIFYING INPUT DATA WITHOUT PROGRAM MODIFICATION

Therefore, namelist-like inputs, rigid format input and special input procedures can be accommodated by the general input modification. For example, in a true namelist input, a data base variable would be entered as follows:

```
NAM1 = 'V1',
```

NAM1 is the name of the NAMELIST variable. V1 is the data base name. The delimiters specify the field width to be employed in replacing the data base name with the corresponding data base value. Similarly for a formatted input where all that normally appears on a card is a number the data base input procedure is simply:

```
'V1'
```

The delimiters are placed at the appropriate card columns defining the field for the data element.

Additional capability is available when namelist input is used by the applications program. Entire arrays may be transferred to the input stream.

```
NAME - 'VARRAY'
```

where VARRAY is a data base array. If the data in VARRAY is more than three elements, additional 'cards' are created to pass all the information in the VARRAY to the input stream of the applications program.

3.2.3.3 Combining Data Base Information in the Modified Input Stream. - Data base variables and constants may be combined much like the capability described for the ADD command in Section 3.2.2.2.

For example, the operation may be performed:

```
'V1 * V2'
```

The above example illustrates how the multiplication of data base variable V1 by data base variable V2 can be performed prior to replacing the delimiter set with the product of the multiplication. A new data base variable representing the combination is not created. The product never resides in the data base, only in the modified input stream. V1 must be a data base variable but V2 may be a data base variable or constant.

The operation illustrated above indicates a multiplication of the two numbers enclosed in delimiters prior to replacement of the delimited command with the product. Any algebraic operator may be employed.

- + addition
- substraction
- \* multiplication
- / division
- \*\* exponentiation

More than one of the above operations may be performed within the delimiters.

'V1 + V2 \* V3'

Up to ten operations may be performed within a single command. The hierarchy of operations is the same as that described for the ADD command.

In general, array elements may be used in the replacement command:

'V1(5)'

or

'V1(5) \* V2(6)'

One exclusion from this capability is the first element of an array:

'V1(1)'

The above illustration is not an acceptable statement to the DIALOG executive for transferring the first element. Any other element of an array may be employed. A convenient means of avoiding the above limitation is the following two cards:

'ADD NEW = V1(1)'

'NEW'

The ADD command defines a new location which will contain V1(1) the replacement command will place the variable NEW (i.e. V1(1)) in the modified input stream.

A special feature of the replacement command is the element-by-element combining of entire arrays with constants. As an illustration, consider the example:

`'V1 * V2'`

where the data base variables V1 and V2 are arrays. The above command specifies the element-by-element multiplication of the arrays. If one array has fewer elements than the other, the combining of elements ceases after the shorter array is exhausted and the rest of the longer array remains unchanged. The variable V2 may be a constant or data base name.

3.2.4 Communicating Information from the Applications Programs to the Data Base. - The communication of information from an applications program to the data base generally, but not always, requires modification of that applications program. This modification is usually trivial and involves little programming knowledge to accomplish. Further, a mechanism is available within the FORTRAN language which further simplifies the task. This mechanism is the NAMELIST output feature. A more detailed description of the modification procedure is described in Section 4. The objective of the modification is to generate a special file of information available to the DIALOG executive system, which contains the desired information in the proper format.

The special file is interrogated by DIALOG for name oriented information in the following format:

`$name name = value, value,$`

Note the similarity between the above format and the FORTRAN NAMELIST feature.

In generating the file within the applications program, the analyst has the option of using NAMELIST (in FORTRAN programs only) or simply simulating NAMELIST by following four basic rules.

1. Separate name and values with equal (=) signs  
(N = V1).
2. Separate name/value sets with commas (,)  
(N1 = V1, N2 = V2).

3. Separate values (as in an array) with commas (,)  
(N3 = V1, V2).

4. Delimit multiple name value sets with a delimited  
ADD like "command" (\$name.....\$).

Once the coding for placing the desired information on the special file is completed, and the modified program is stored, the modified program can be used within the DIALOG executive system.

#### 4.0 INSTALLATION OF THE DIALOG EXECUTIVE SYSTEM ON A TYPICAL 1100 SERIES COMPUTER

The DIALOG executive system can be installed on any Univac computer which has an operating system containing control card linking capability. The linking capability is provided by an operating system utility program which directs the system to read from a user specified file other than the input file.

For computer installations not having this capability, a system modification must be performed. At Johnson Space Center (JSC), the control card linking utility installed is called ADD. Similar utilities are available at other installations where the DIALOG executive system is in use.

The description of the installation of the DIALOG executive system is strongly dependent upon the computer operating system in use. Each computer complex has unique operating system features not generally available at other computer complexes. System differences are reflected in the control cards.

As a specific example, the discussion in this section is directed toward the installation of the DIALOG executive system at Johnson Space Center. Therefore, the control cards described (including ADD) will be peculiar to the Univac 1110 Exec VIII operating system employed at the JSC facility. The DIALOG executive system is also available on the Univac 1108 Exec II system at JSC but the example illustrated in this section is for the 1110. Complete detailed information for control card usage at JSC may be obtained from the literature available upon request from the JSC computer complex.

Before installing the DIALOG executive system the storage devices for the independent programs must be chosen. This affects both the procedure in storing the program library as well as the entries into the control card data base. The program storage devices can be either disk (online) or tape (offline) at JSC. The two device types may be mixed if so desired. However, the disk is the most commonly used at JSC and considering the number of programs involved, the use of disk assures a

smoother operating system. Therefore, disk storage will be discussed exclusively in this section.

In general terms the installation of the DIALOG executive system involves four basic tasks:

1. Compile and store the DIALOG executive programs.
2. Compile and store a library of independent programs.
3. Create a control card data base containing the control card sequences of each independent program.
4. Store the DIALOG executive system including the control card data base.

When all of the above tasks have been accomplished, the use of the DIALOG executive system may commence as described in Section 5. The above tasks will be described in detail in this section.

#### 4.1 Compilation and Storage of the DIALOG Executive Program

The objective of this paragraph is to describe first the compilation and storage of DIALOG, a brief description of how DIALOG fits into the DIALOG executive system, a discussion of data base parameters and finally a deck set up for storing DIALOG.

The DIALOG executive has two main programs:

DBINIT - initialization

DIALOG - language processing

DBINIT is the DIALOG executive system initialization program. DBINIT is called only once when a new design data base (see Section 5) is being created, but performs no language processing function. DIALOG is the language processing program. DIALOG is executed initially to establish the execution sequence of the independent programs; then is called after the execution of each independent program to process the data for the independent programs.

4.1.1 Data Base Parameters. - Before compiling the executive programs the design data base size must be established through the data base parameters. These parameters control the number of names and number of data elements which can be stored in the dynamic data base as described in Section 3. They also control the length of the description for each entry. All data base parameters directly affect the core storage requirements for the DIALOG executive system.

As an example, for a data base size of 3000 data elements, 200 names and a description length of three computer words, the core storage requirement for the system is 55000 (base 8). The above parameters are the standard values for the DIALOG executive system. Unless there are specific requirements for the data base parameters, they may be left as they are. However, they are usually set such that the core size matches the core size of the largest independent program anticipated for the system.

The alteration of the data base parameters requires changes to the following programs or subroutines:

- DBINIT - data base initialization program
- DIALOG - language processing program
- DBLOAD - data base loader subroutine
- INITL - DIALOG initialization subroutine

The data base consists of two parts, a packed array for storage of data elements and a directory of names and pointers to the data element locations. Both the number of names and number of data base elements may be altered. The directory also includes provisions for a description array for each name. The data base parameters involved in changing the data base size are:

- DBLEN - length of the data base (i.e. the packed array of data elements)
- DIRLEN - directory length or number of name entries possible in the directory
- DIRWID - directory width

The directory width consists of the following elements or words:

1. One word for the name.
2. One word for the value (pointer).
3. One word for the "hash" table.
4. One word for the "collision" table.
5. One word for the update command.
6. A variable number of elements for the description.

The minimum directory width is five which allows no space for descriptive information in the directory. Therefore, the variable DIRWID may be computed as follows:

DIRWID = 5 + number of words of description

Once the three data base parameters are established, the parameters and dimensions may be set in the four affected routines:

DBINIT:

```
COMMON /DTBASE/ IDATA(2*DBLEN)
COMMON /DIRECT/ IOPR(50), INUM(60), ID(DIRLEN*DIRWID)
```

DIALOG:

```
COMMON /DTBASE/ IDATA(2*DBLEN)
COMMON /DIRECT/ IOPR(50), INUM(60), ID(DIRLEN*DIRWID)
```

DBLOAD:

```
COMMON /DTBASE/ IDATA(2*DBLEN)
COMMON /DIRECT/ IOPR(50), INUM(60), ID(DIRLEN*DIRWID)
```

INITL:

```
DATA KEYLEN /1/, DBLEN /DBLEN/, DIRLEN /DIRLEN/,
DIRWID /DIRWID/
```

The underlined parameters are the only ones requiring alteration by the user.

4.1.2 Deck Setup for DIALOG Storage. - The DIALOG executive programs are stored in source form and relocatable binary elements (LGO file) on tape and in absolute element form on disk. The control cards, excluding the "accounting" cards, are discussed in Appendix A. It is assumed the reader is familiar with the above mentioned "accounting" cards. (See JSC computer usage manuals for complete details.)

The deck setup for compiling and storing DIALOG and DBINIT is shown in figure 4-1. This deck setup assumes space on disk has been preassigned.

The sequence of utility operations is:

- Compile the source code.
- Store the source and relocatable binary on tape
- Map the absolute element program.
- Store the absolute elements on disk.

Source and relocatable binary are stored on a tape using the system utility COPOUT. All programs stored on tape must have a "label" supplied by the JSC computer programmer. This label is placed on the tape by the systems operator. All disk storage must be preassigned by the computer operations personnel before information can be secured on permanent file.

## 4.2 Compilation and Storage of a Library of Programs

Since the DIALOG executive controls the sequence of execution of a library of independent programs, each program in the library must be stored in the computer system. The library includes not only user supplied programs but also system programs such as the compiler and the data base storage and retrieval programs.

The manner in which the programs are stored is analogous to the storage of the DIALOG program depicted in figure 4-1. However, there are two additional points to consider in preparing an analysis program for use with the DIALOG executive system:

```

@FOR,IS DBINIT,DBINIT
      (SOURCE CODE)
@FOR,IS DIALOG,DIALOG
      (SOURCE CODE)
/
/
/
/
/
@FOR,IS WRTUE,WRTUE          (LAST SUBROUTINE)
      (SOURCE CODE)
@ASG,T DLOG,8C,A00000
@COPOUT TPF$,,DLOG.
@MAP ODLOG,ODLOG
      INSERT DIALOG
@MAP ODBIN,ODBIN
      INSERT DBINIT
@ASG,A ODIN
@COPY ODLOG,ODIN,DLOG
@COPY ODBIN,ODIN,DBIN

```

FIGURE 4-1 DECK SETUP FOR INITIALLY STORING DIALOG

1. The program may require modification to provide a special output file of data base information as briefly mentioned in Section 3.
2. The program may be stored in absolute element form for high speed loading and reduced core requirements.

4.2.1 Program Modification to Provide Data Base Information. - The communication of information from an applications program to the data base generally, but not always, requires modification of that applications program. There are a number of analysis programs available which generate files of information suitable for use with the DIALOG executive system, either directly or through an interface program. An interface program is often the most convenient means of extracting data from an analysis program.

The interface program obtains its input from a file generated by another program. The output will be precisely as specified in the following paragraphs. In this sense, the interface program is simply another analysis program in the program library.

Whether the original program is modified or an interface program is written, the generation of the data base output file is usually trivial and involves little programming knowledge to accomplish. If FORTRAN is the program language, a mechanism is available which further simplifies the task. This mechanism is the NAMELIST output feature of FORTRAN. The use of NAMELIST is briefly discussed below.

4.2.1.1 Creating a Special Output File - The objective of the modification is to generate a special file of information available to the DIALOG executive system, which contains the desired information. This is accomplished with an additional file.

```
WRITE (14,- - - - -
```

In the above illustration, the file unit 14 is the special file which is read by the DIALOG executive program. The file number is optional and may be any available unit. However, unit 14 has been adopted as the standard on the 1100 series computers. The information placed in this file is interpreted and then transferred to the data base by the DIALOG executive program. The mechanism by which the file is passed from the applications program to the DIALOG executive, is referred to as file substitution. The file substitution is accomplished with the control cards stored in the control card data base. As an illustration of file substitution technique, consider the execution of PGMA above. Prior to the execution control card for PGMA would be:

```
@USE 14, NMLIST
```

The above card specifies that unit 14 will be the system file name NMLIST. These files are addressed internal to PGMA by the logical unit number 14, but addressed externally by the system file name, NMLIST. They are sometimes referred to as internal and external names respectively. DIALOG recognizes NMLIST as a file containing potential data base information.

4.2.1.2 Format of the Special Output File. - NMLIST is interregated by DIALOG for name oriented information in the following format:

```
$name name = value, value, $
```

Note the similarity between the above format and the ADD command described in Section 3.2.1. Also it is identical with the FORTRAN NAMELIST feature.

In generating the NMLIST file within the applications program, the analyst has the option of using NAMELIST (in FORTRAN programs only) or simply simulating NAMELIST as discussed in Section 3.

Generally the name selected is one which is similar to the program name such as:

```
$PGMAO
```

for PGMA output. The advantage of using this naming convention is apparent in the actual data base construction. The name PGMAO is stored in the data base identifying which program or "command" which last updated the value or values stored.

4.2.1.3 Use of the NAMELIST Feature in FORTRAN. - If the applications program is written in FORTRAN, the analyst can use the NAMELIST feature to write the special data base output file. For example, to transfer the variables ANAME, BNAME, CNAME I1, I2, JNAME and associated values to the data base, the following modification to PGMA is required at or near the exit point:

```
NAMELIST/PGMAO/ANAME,BNAME,CNAME,I1,I2,JNAME
```

```
WRITE (14,PGMAO)
```

The format of unit 14 for the above illustration is shown in figure 4-2. The DIALOG executive program interrogates unit 14 after the execution of the applications program to find variable names and values to be entered into the data base.

4.2.2 Storage of an Absolute Element Program. - In performing a given analysis, the applications program user usually compiles and loads the relocatable binary elements. These operations require the use of the operating system compiler, FOR and the operating system loader, XQT in the following manner:

```
@FOR,MAIN
```

```
@XQT
```

The standard loader must reside in core while it loads the program. Further, the loader requires an additional space allocation for "loader tables." The additional space requirements depend upon the particular program involved, but the total space for loader and tables varies from 5000 (base 8) to 15000 (base 8). The space required by the loader must be added to the space required by the resulting applications programs. The total space for both the loader and program is the central memory field length required for the job. The field length is automatically reduced after the program is loaded (i.e. the loader space becomes available to the system for other uses).

The above method of analysis results in two disadvantages:

1. More central processing and peripheral processing time used to compile and/or load.

```
$PGMAO  
  
  ANAME = VALUE,  
  
  BNAME = VALUE,  
  
  CNAME = VALUE,  
  
  I1    = VALUE,  
  
  I2    = VALUE,  
  
  JNAME = VALUE,  
  
$END
```

FIGURE 4-2 FORMAT OF THE NMLIST OUTPUT FILE

2. Job turn around suffers from the increased field length requirements.

The disadvantages illustrated above both result in reductions in efficiency and productivity of both computers and analysts. In the DIALOG executive system, most programs are compiled and stored as absolute elements. An absolute element program is one generated by the MAP program. The absolute element program is a self contained program which includes all the system routines called for by that particular program. The advantages of storing the absolute element program are that it:

1. Reduces the field length requirement for the program by the size of the loader and loader tables.
2. Considerably reduces the load time required for the program. All external references are resolved. This means all operating system routines called by the program are stored in the absolute element.

One disadvantage of storing an absolute element program is the disk storage requirements are increased by the space required for system routines. This is not considered to be a serious disadvantage. Another disadvantage is that updating of an absolute element program requires remapping of the entire binary program. The above are not considered major disadvantages.

#### 4.3 Assembly of the Control Card Data Base

The control card data base, CCDATA, contains all the control card sequences required to retrieve and execute the library of independent programs. After the independent program is stored and prior to creation of the DIALOG executive system, the control card sequences must be assembled. It is also desirable to assemble a series of utility procedures for the purpose of data disposition. These procedures enhance the usability of the DIALOG executive system.

The creation of a control card data base is a function of the DIALOG executive. DIALOG reads the control card

sequences from input cards and stores them in CCDATA. The delimited control card directive which creates the control card data base is:

'INITAL CCDATA'

One and only one space may appear between the words INITAL and CCDATA. This directive is followed by a file of information containing the control card sequence for the various applications programs and utility functions as illustrated in figure 4-3. The first entry name must have an opening delimiter immediately before the name. A closing delimiter must appear after the last entry on a separate card as shown. The file must be terminated by a \$EF\$ card.

The following paragraphs describe the formation of a single entry in the control card data base for a hypothetical applications program.

4.3.1 Construction of a Control Card Sequence for a Data Base Entry - An illustration of a data base entry is shown in figure 4-4. The essential features are:

- a. A name for the control card sequence followed by an equal (=) sign.
- b. A retrieval card is the first card in the sequence.
- c. An execution card.

The name is selected by the analyst at the time the control card entry is constructed. The name will be used to retrieve the control card sequence. The EXECUTE directive is used for this purpose.

'EXECUTE PGMA'

Common practice dictates the use of the acronym associated with an applications program such as AESOP for Automated Engineering and Analysis Program. Most applications programs have such an acronym associated with them.

Application program execution sequences require an @ASG card first. If no application program is being used in the execution sequence, the first card may be a:

@LABEL:

where LABEL may be any label made up of 1-6 characters.

'CREATE CCDATA'

'PGMA =

{  
CONTROL  
CARD  
SEQUENCE  
}

PGMB =

{  
CONTROL  
CARD  
SEQUENCE  
}

PGMC =

{  
CONTROL  
CARD  
SEQUENCE  
}

\$EF\$

FIGURE 4-3 ILLUSTRATION OF A CONTROL CARD DATA BASE  
CREATION FILE

PGMA =

@ASG,A EX42-00002\*ODIN-PGMA

- - - -

(OTHER PRE-EXECUTION CONTROL CARDS)

- - - -

@USE 14,NMLIST

@EX42-00002\*ODIN-PGMA

- - - -

(POST-EXECUTION POST CARDS)

- - - -

FIGURE 4-4 SAMPLE CCDATA ENTRY

Following the @ASG card any utility functions required by the applications program prior to execution may be specified. One example could be a USE for some file needed by the applications program. The execution card begins with the file name from the @ASG card.

#### 4.4 Storage of the DIALOG Executive System

The final task in the installation of the DIALOG executive system is the storage of the executive programs and special procedures. When this task is accomplished, simulations can begin. The DIALOG executive system consists of two FORTRAN programs discussed in Section 4.1, the control card data base discussed in Section 4.2 and 4.3 and the initialization procedure shown in figure 4-5.

DIALOG executive system initialization procedure

DIALOG executive programs

Control card Data Base, CCDATA

The entire system is stored on a disk as source or binary records. The independent procedure is stored as source. The purpose is two fold.

1. Minimize the number of operating system control cards required to start a simulation.
2. Provide a convenient means for modification of the initialization procedure (as in the use of a restart capability).

Two control cards (other than JOB and USER cards) are required to start a simulation:

@ASG,A EX42-00002\*ODIN-LUI

@ADD EX42-00002\*ODIN-LUI

The ODIN procedure then 'boot straps' the rest of the DIALOG executive system in from the disk storage. Modification of the ODIN procedure may be accomplished by the usual methods at the JSC computer complex.

```

@FREE TPF$.
@ASG,T TPF$,F/50//1000
@DELETE,C NMLIST
@ASG,CP NMLIST,F2
@FREE NMLIST
@ASG,A NMLIST
@DELETE,C LUSOP
@ASG,C LUSOP,F2
@FREE LUSOP
@ASG,A LUSOP
@DELETE,C COPY5
@ASG,C COPY5
@FREE COPY5
@ASG,A COPY5
@DELETE,C DBASE
@ASG,CP DBASE,F2/50/TRK/100
@FREE DBASE
@ASG,A DBASE
@QUAL B
@ASG,A *MODIN
@FREE,D *MODIN
@ASG,CP *MODIN
@FREE *MODIN
@ASG,A *MODIN
@QUAL A
@ASG,A *MODIN
@FREE,D *MODIN
@ASG,CP *MODIN
@FREE *MODIN
@ASG,A *MODIN
@DELETE,C CCDATA

```

FIGURE 4-5A DIALOG EXECUTIVE SYSTEM EXECUTION  
INITIALIZATION PROCEDURE.

```

@ASG,C CCDATA,F2/50/TRK/100
@FREE CCDATA
@ASG,A CCDATA
@DELETE,C CONTRL
@ASG,CP CONTRL,F2/5/TRK/10
@FREE CONTRL
@ASG,A CONTRL
@USE 14,NMLIST.
@USE 25,DBASE.
@QUAL EX42-00002
@USE T,*ODIN-DIALOG
@T.ODBIN
@USE 22,*ODIN-LUI.
@USE 23,LUSOP.
@USE 24,COPY5.
@USE 25,DBASE.
@QUAL A
@USE 27,*MODIN.
@USE 28,CCDATA.
@USE 29,CONTRL.
@T.ODLOG
@FREE EX42-00002*ODIN-DIALOG
@FREE *MODIN
@ADD,P *MODIN.
@FREE *MODIN
@ADD,P *MODIN.
.
.
.
@FREE *MODIN
@ADD,P *MODIN

```

FIGURE 4-5B DIALOG EXECUTIVE SYSTEM EXECUTION  
INITIALIZATION PROCEDURE.

## 5.0 USE OF THE DIALOG EXECUTIVE SYSTEM

The discussion in Section 4 centered around the installation of the DIALOG executive system including a control card data base. The present discussion deals with the use of the DIALOG executive system with a library of independent applications programs. Figure 5-1 illustrates how the DIALOG executive system is set up and used. There are four basic steps in using the system. First the task must be defined, which includes a consideration of the technology areas to be included in the analysis. Further the depth of analysis in each technology area must be selected. The depth of analysis and subsequent program selection will have a direct bearing on the computer resources which will be required. The second task is the selection of the applications programs and the sequence which will be executed. This requires some understanding of the basic data required by each program and the information each program generates. The above tasks may be a team effort if the simulation is large and/or requires many programs.

A survey of the existing ODIN library programs will aid in the selection process. If a suitable set of programs is not available, one of the following procedures may be employed:

1. Locate the necessary programs from an outside source.
2. Develop the necessary programs to serve the purpose.

In either of the two above cases, the program modification for use with the DIALOG executive system as discussed in Section 4 may be required.

The third task is the definition of the interprogram data which will be stored in the data base. Included in this definition are the study parameters, those elements of data which drive the study either through a parametric variation or an optimization process. Additionally, the performance criteria must be defined. The performance criteria is the desired study output information such as the weight or cost of the system under study. Often there are study constraints, those conditions which must not be violated in an acceptable solution. The study constraints and performance criteria are the basic information used in guiding the selection of values for the study parameters.

1. DEFINE THE SIMULATION TASK
  - TECHNOLOGY AREAS TO BE CONSIDERED
  - DEPTH OF ANALYSIS
2. SELECT THE APPLICATIONS PROGRAM SEQUENCE
  - SURVEY THE AVAILABLE PROGRAMS
  - UPDATE THE PROGRAM LIBRARY
  - DEFINE THE EXECUTION SEQUENCE
3. DEFINE THE DATA BASE INFORMATION
  - STUDY PARAMETERS
  - PERFORMANCE CRITERIA
  - STUDY CONSTRAINTS
  - LIBRARY INTERPROGRAM DATA
  - MONITORING INFORMATION
4. DECK SETUP
  - SETUP NORMAL DATA FOR ALL PROGRAMS
  - INSERT THE CONTROL DIRECTIVES
  - INSERT THE COMMUNICATION COMMANDS

FIGURE 5-1 PROCEDURE FOR THE USE OF THE DIALOG EXECUTIVE SYSTEM

Interprogram data must also be defined in the data base. These data include intermediate results produced by one applications program which are used by other applications programs. Finally the data base may include information which may be used for monitoring the study.

The fourth and final task is the actual deck setup. Deck setup is the task which is addressed in this section. The implication is that the other three tasks have been performed; the remaining objectives are to set up the normal data for the selected applications programs then insert the proper control directives and communication commands to perform the desired simulation. This section is divided into discussions of control directives and communication commands. Control directives are instructions to DIALOG for the control of the sequence of execution of the independent applications programs. Communication commands are instructions to DIALOG for the merging of data base information with applications program data. Finally, the use of standard utility procedures will be discussed.

## 5.1 Control Directives

The use of the DIALOG executive system requires that the DIALOG executive programs and procedures be available on disk before the DIALOG control directives can be employed. Figure 5-2 is an illustration of the operating system control cards at JSC required to access the DIALOG executive system. Following the four control cards shown, all remaining information is processed by the DIALOG executive. As such the rules governing the use of the DIALOG executive system apply.

The general arrangement of the input data to the DIALOG executive system is shown in Figure 5-3. Here a hypothetical simulation is setup which illustrates the use of all control directives and alternates. Each control directive in this illustration will be described in detail. A summary of control directives is given in Appendix A. Generally speaking, the control directives have the following format:

'directive name'

The directive is enclosed by DIALOG delimiters (''). No space is permitted between the first delimiter and the directive. One and only one space is permitted between the directive and the name. The delimiter for the JSC system is a 2-8 punch.

@RUN - - - -

@QUAL EX42-00002

@ASG,A \*ODIN-LUI

@ADD \*ODIN-LUI

{ DIALOG  
CONTROL  
DIRECTIVES }

DIALOG  
INPUT

@FIN

NOTE: \*ODIN-LUI CONTAINS THE ODIN INITIALIZATION  
PROCEDURE.

FIGURE 5-2 SYSTEM CONTROL CARDS REQUIRED TO ACCESS  
THE DIALOG EXECUTIVE SYSTEM.

```

                                'RESTART'
'INITAL DBASE'                OR      'UPDATE DBASE'
    [FILE OF DATA INPUT TO THE DESIGN DATA BASE]
$EF$

'UPDATE CCDATA'                OR      'INITAL CCDATA'
    [FILE OF DATA INPUT TO THE CENTRAL CARD DATA BASE]
$EF$

'DESIGN START'                (OPTIONAL DESIGN IDENTIFIER)
'EXECUTE PGMA'
    [FILE OF INPUT DATA FOR PGMA]
$EF$

'LOOP TO START'                (CONDITIONAL BRANCHING LOGIC)
'IF V1, LT, V2'
$EF$

'EXECUTE PGMB'
    [FILE OF INPUT DATA FOR PGMB]
$EF$

'PRINT DBASE'                (OPTIONAL PRINT COMMAND)
'END'
@FIN

```

FIGURE 5-3      EXAMPLE OF A HYPOTHETICAL EXECUTION SEQUENCE  
ILLUSTRATING THE USE OF ALL CONTROL DIRECTIVES.

5.1.1 INITIAL Directive. - Initially the INITIAL directive is used to establish a design data base.

'INITAL DBASE'

[ file of data  
to initially  
establish DBASE ]

\$EF\$

The INITIAL directive is followed by a file of data to initially establish the design data base. Initial data is not essential to the operation of the system as data may be added at any point in the execution sequence. However, the \$EF\$ end-of-file mark is required regardless of whether data is entered. Data is always entered via the communication commands described in Section 5.2.

The INITIAL directive is also used to create a control card data base.

'INITAL CCDATA'

[ File of control  
card data base  
entries ]

NOTE: Control cards start in  
column 7 (shifted by  
one computer word)

\$EF\$

The control directive illustrated above creates a new control card data base, overwriting any existing one. The creation of a control card data base is described in detail in Section 4. Usually this directive is not used in a routine simulation since a control card data base already exists.

5.1.2 RESTART Directive. - The INITIAL DBASE directive must be the first directive in the sequence unless the simulation is the continuation of an earlier run. In the latter case, the following directive is used in place of INITIAL DBASE:

'RESTART'

For the illustrated command to be effective, the data base created in the earlier run must have been saved by use by the ENDODN procedure described below. Further, the DIALOG executive system initialization procedure differs in case

of a restart. Figure 5-4 illustrates the restart procedure. Here the execution of DBINIT is replaced by a @ASG control card for the retrieval of the previous design data base.

5.1.3 UPDATE Directive. - The UPDATE directive is used for updating existing information in the data base and has the following format:

```
'UPDATE name'  
    [ file of data  
      to update existing  
      data base ]  
$EF$
```

The name associated with the illustrated directive can be DBASE or CCDATA. The directive may be used for DBASE after a RESTART directive for updating information in an existing design data base. Often the UPDATE directive is used with CCDATA to alter an applications program procedure. A card-by-card update may be performed in the following manner:

```
'UPDATE CCDATA'
```

Following this control directive, the analyst lists the desired modifications.

```
'PGMA =  
    [ new control  
      card  
      sequence ]  
,
```

The above example illustrates the complete replacement of a control card sequence. If, however, the analyst desired to change only one card in the sequence, the following cards must be employed:

```
'PGMA (3) =  
    [ replacement control card ]  
,
```

The above example illustrates the replacement of the third card in the sequence. The third and fourth card could be replaced in the following manner:

```
@RUN - - - -  
  
@QUAL EX42-00002  
  
@ASG,A *ODIN-LUI  
  
@ASG,T RESTART,F  
  
@DATA *ODIN-LUI, RESTART  
  
-43,43  
  
@ASG,A OLDATA,      (STORED DATA BASE)  
  
@COPY OLDATA,DBASE  
  
@END  
  
@ADD RESTART
```

FIGURE 5-4 ILLUSTRATION OF THE RESTART PROCEDURE.

'PGMA (3) =

[ replacement for third card  
replacement for fourth card ]

The requirement for updating the control card data base stems from the fact the library programs are often modified or replaced as a result of revisions or new versions. Often more than one version of the same program exists. This is particularly true during periods of program development. During these periods of development, the analyst may wish to evaluate a test version of a program in the DIALOG executive system. The UPDATE directive affords the individual analyst the opportunity to make temporary control card modifications to the DIALOG executive system without affecting the other users of the system.

5.1.4 DESIGN Directive. - The DESIGN directive is used to establish a point in the execution sequence to which control may be returned (or skipped to) via a LOOP TO directive described below. The DESIGN directive is analogous to a statement label in a FORTRAN program. The format of the DESIGN directive is:

'DESIGN name'

The name may be any name up to 6 characters but must not duplicate an existing data base name.

5.1.5 EXECUTE Directive. - The EXECUTE directive is used to execute a sequence of control cards from the control card data base. The format is:

'EXECUTE name'

[ data file for the  
applications program ]

\$EF\$

The name may be any name assigned to a control card sequence in the control card data base. This is the basic command for the execution of applications programs. The data file is the data for the applications program involved. The \$EF\$ card is required even if no data is involved with the execution.

5.1.6 LOOP TO Directive. - The LOOP TO directive is used to transfer control to another point in the execution sequence.

```
'LOOP TO name'
```

In the above illustration, control is transferred to name. Name must be the name associated with a DESIGN directive described in Section 5.1.4.

5.1.7 IF Directive. - The IF directive is a conditional branching directive used in conjunction with the LOOP TO directive in the following manner:

```
'DESIGN START'  
- - - - -  
- - - - -  
'LOOP TO START'  
  
'IF V1 . LT . V2'  
  
$EF$  
- - - - -  
- - - - -
```

In the above illustration, control is transferred to START if V1 is less than V2. V1 and V2 are design data base variables or constants. Multiple IF directives may be associated with any LOOP TO directive. When multiple IF directives are used, any condition specified will trigger the LOOP directive. If no conditional IF directives are used, the LOOP TO directive is mandatory. A \$EF\$ card is recommended after OP TO directive sets.

5.1.8 PRINT Directive. - The PRINT directive is used to print the design and control card data bases DBASE and CCDATA. The format is:

```
'PRINT DBASE'
```

or

```
'PRINT CCDATA'
```

This command may be placed ahead of an execution directive. It cannot be intermixed with a file or applications program data.

5.1.9 END Directive. - The END directive is used to signify the end of a simulation. It has the following format:

'END '

All simulations must be terminated with the END directive. A summary of the control directives is given in Appendix A.

## 5.2 Communication Commands

The communication commands provide a means of transferring information to and from the data base. The following types of transfer are included:

1. Transfer of information from the analyst to the data base.
2. Transfer of information from the data base to the applications program.
3. Transfer of information from the applications programs to the data base.

In the transfer of information from the analyst to the data base, three commands are used:

1. ADD command for adding or updating information in the data base.
2. DEFINE command for defining data base variables and reserving space in the data base.
3. . (comment) command for identifying applications program data.

In the transfer of information from the data base to the applications program, the basic command is the replacement command for replacing data base names and data base name/value combinations with values from the data base.

In the transfer of information from the applications program to the data base, a special extension of the ADD command is employed. The "ADD-like" command is used in generating a special output file which can be placed into the data base.

Each of the above commands will be described in detail in this section. Examples covering most objectives will be presented.

5.2.1 The ADD Command. - The ADD command is the basic communication command available to the analyst for adding or updating information in the data base. The general format of the command is:

```
'ADD name 1 = value 1, name 2 = value 2,  
      name 3 = value 3, value 5, value 5,  
      value 6, name 4 = value 7,'
```

It may be used in creating data in the data base as well as for modifying the data base at any point in the execution sequence. The ADD command must appear within the file of information following one of the control directives:

'INITIAL DBASE'

'UPDATE DBASE'

'EXECUTE name'

The ADD command may be used for adding any type of information to the data base (i.e. real, integer, hollerith or logical) the data type being determined by the actual data entry. The information may be a single element or an entire array of information. Arrays may be of mixed types under certain circumstances. Combinations of data base variables and constants using the five common operators (+, -, \*, /, \*\*) may also be employed within the ADD command. However, mixed mode arithmetic is not permitted in combining data base variables. Specific rules and exceptions will be presented in the examples below.

The following list of rules of syntax or pattern of construction apply to the ADD command:

1. The opening delimiter (' or 2-8 punch) may be in any column.
2. No spaces may appear in the character string 'ADD (including the delimiter).
3. One or more spaces must appear between the ADD and the first name.
4. One or more name = value combinations may appear on each card.

5. The 'ADD command may be continued from card to card. Any number of cards may be used.
6. A card must be terminated with a comma (,) or a DIALOG delimiter (').
7. A continuation card may start with a name or value.
8. The 'ADD command must be terminated with a DIALOG delimiter (').
9. The closing delimiter may be on a separate card.
10. The last comma (,) before the closing delimiter is optional.
11. The maximum number of names and values (including the command) is 20 per card.

The most common SYNTAX errors result from failure to comply with rules 2, 6 and 8. Attention is specifically drawn to these rules.

The following pages present some examples. Each example is discussed. In addition, a sample data base printout for all examples will be given. Discussion of the examples is presented in the following format:

OBJECTIVE:	A concise statement of the analysis goal to be achieved by the command.
SYNTAX:	Pattern of formation of the command.
EXAMPLES:	A list of one or more examples. Sometimes incorrect examples are given and are noted as ILLEGAL. In these cases the results are either incorrect or unpredictable for the objective stated.
RESULT:	A brief description of the data base entry which will result from the use of the exemplified command.
RESTRICTIONS:	In some cases restrictions in addition to these described above will be given.

All of the ADD command examples presented in this section are illustrated in figure 5-5. These examples are listed from cards which were actually processed by the DIALOG executive program. The delimiter (≠) shown in the illustration results from the particular character set employed by the printing device. The data base entries which resulted from the examples in figure 5-5 are illustrated in figure 5-6.

#### 5.2.1.1 Adding Fixed Element Information. -

OBJECTIVE	To add or update elements of information to the data base.
SYNTAX:	'ADD name = value'
EXAMPLES:	'ADD A = 10.' 'ADD B = 2' 'ADD C = ALPHA' 'ADD D = .TRUE.'
RESULT:	Four data base locations will be created or updated in the data base. Each of the four basic data types are represented. The data type is determined from the actual number. There is no implicit type assumed from the data base name (as in FORTRAN).
ADDITIONAL RESTRICTIONS:	1. Hollerith information and logical variables which are to be stored in the data base cannot be data base names.

#### 5.2.1.2 Adding Multiple Data Elements. -

OBJECTIVE:	To add or update a series of elements to the data base by a single ADD command.
SYNTAX:	'ADD name 1 = value 1, name 2 = value 2'
EXAMPLES:	'ADD A = 10., B = 2, C = ALPHA,            See Restrictions D = .TRUE.'

# ADD STATEMENT EXAMPLES #

```
#ADD A = 10. #
#ADD B = 2 #
#ADD
      C = ALPHA #
#ADD D = .TRUE. #
#ADD E = A #
#ADD F = B #
#ADD G = A * 2. #
#ADD H = B + 3 #
#ADD I = A + G * 2. #
#ADD J = 10., 20., 30. #
#ADD K = 2, 4, 6 #
      L = ALPHA, BETA, GAMMA #
#ADD
      M = .TRUE., 3., 9.,
#
#ADD N = 2*3., 4., 9. #
#ADD O = 2, 5., 6. #
#ADD P = J(1) #
#ADD Q = K(2) #
#ADD R = A * O(2) #
#ADD S = J(3) + Q(2) #
```

NOTE: THE DELIMITER (,) RESULTS FROM THE LISTING EQUIPMENT USED.

FIGURE 5-5 ADD COMMAND SUMMARY

NAME	LOCATION	DIMENSION	CURRENT VALUE(S)	ORIGIN	DESCRIPTION
A	1	1	10.	ADD	REAL VALUE
B	3	1	2	ADD	INTEGER VALUE
C	5	1	ALPHA	ADD	HOLLERITH VALUE
D	7	1	.TRUE.	ADD	LOGICAL VALUE
E	9	1	10.0000000000000000	ADD	E=A
F	11	1	2	ADD	F=B
G	13	1	20.0000000000000000	ADD	G=A*2.
H	15	1	5	ADD	H=B+3
I	17	1	60.0000000000000000	ADD	I=A+G*2.
J	19	3	10. 20. 30.	ADD	REAL ARRAY
X	25	3	2 4 6	ADD	INTEGER ARRAY
L	31	3	ALPHA BETA GAMMA	ADD	HOLLERITH ARRAY
M	37	3	.TRUE. 3 9.	ADD	MIXED LOG-INT-REAL ARRAY
N	43	4	8.0000000000000000 8.0000000000000000 9. 9.	ADD	SHORTHAND LOAD
O	51	3	2 5. 6.	ADD	MIXED TYPE ARRAY
P	57	1	10.0000000000000000	ADD	P=J(1)
Q	59	1	4	ADD	Q=K(2)
R	61	1	60.0000000000000000	ADD	R=A*O(3)
S	63	1	35.0000000000000000	ADD	S=J(3)+O(2)

FIGURE 5-6 DATA BASE ENTRIES RESULTING FROM THE EXAMPLE ADD COMMAND.

RESULT: Four data base locations will be created or updated in the data base. The value stored will be those presented by the information to the right of each equal (=) sign.

ADDITIONAL  
RESTRICTIONS: 1. Hollerith information and logical variables which are to be stored in the data base cannot be a data base name.

#### 5.2.1.3 Transferring Data Elements. -

OBJECTIVE: To create or update a data base variable by equating (transferring) one variable name to another.

SYNTAX: 'ADD name = name'

EXAMPLES: 'ADD E = A'

'ADD F = B'

RESULT: The variable E and F are created or updated. The information (real or integer) is transferred from A and B to the new locations for E and F.

ADDITIONAL  
RESTRICTIONS: 1. A and B cannot be hollerith or logical variables.  
2. Neither E or F can be an array name.

#### 5.2.1.4 Combining Data Elements with Constants. -

OBJECTIVE: To create or update a data base variable by combining a data base variable with a constant using an arithmetic operation.

SYNTAX: 'ADD name = name \* value'

EXAMPLE: 'ADD G = A \* 2.' (real)  
'ADD H = B + 3' (integer)  
'ADD G = 2. \* A' (ILLEGAL - See Restrictions.)

RESULT: Two variables, G and H, will be created or updated in the data base whose values will be the combinations indicated. Any of the common arithmetic (+, -, /, \*, \*\*) operations may be performed. Up to ten operations may be performed.

ADDITIONAL

- RESTRICTIONS:
1. The combination of a variable with a constant cannot begin with a constant.
  2. Neither G nor H can be an array name.

5.2.1.5 Combining Data Elements with other Data Elements and Constants. -

OBJECTIVE: To create or update a data base variable by combining other data base variables and constants.

SYNTAX: 'ADD name = name + name \* value'

EXAMPLE: 'ADD I = A + G \* 2'

RESULT: The contents of A will be added to the contents of G. The results of that combination will be multiplied by 2. The result of that combination will be transferred to I. Up to ten operations may be performed.

ADDITIONAL

- RESTRICTIONS:
1. The operations are serial in nature (like a hand calculator). Each operation is performed on the result of the previous operation (s).
  2. The combination of variables must begin with a name.
  3. The name, I cannot be an array name.

#### 5.2.1.6 Adding Arrays. -

OBJECTIVE: To create or update arrays of information in the data base.

SYNTAX: 'ADD name = value, value, value'

EXAMPLES: 'ADD J = 10., 20., 30.,  
K = 2, 4, 6,  
L = ALPHA, BETA, GAMMA, (hollerith  
constants)  
M = .TRUE., 3,5.'

RESULT: Four new or existing arrays will contain the information indicated. Mixed arrays of real and integer values may also be used. Logical arrays (more than one element) cannot be entered.

ADDITIONAL  
RESTRICTIONS:

1. No data base name may appear on the right side of the equation (= sign) as array elements.
2. Logical and hollerith variables may not be mixed with any other type.
3. Hollerith may not be the first entry of an ADD card. (i.e. 'ADD L = ALPHA, BETA, GAMMA,')
4. No more than one logical variable per array may be entered.

#### 5.2.1.7 Adding Constant Arrays. -

OBJECTIVE: To add or update an array of constant information to the data base.

SYNTAX: 'ADD name - n \* value, value'

EXAMPLES: 'ADD N = 2 \* 8., 9., 9.'  
'ADD N = 8., 8., 2 \* 9.' (ILLEGAL)

RESULT: N will be a new or updated array of four elements, each containing values of 8., 8., and 9., 9.

ADDITIONAL

RESTRICTIONS: 1. The integer n must be next to the equal (=) sign. Multiple entries can only be performed on the first n elements.

#### 5.2.1.8 Adding Mixed Arrays. -

OBJECTIVE: To add or update a mixed integer/real array of information.

SYNTAX: 'ADD name = value, value, ...'

EXAMPLE: 'ADD Ø = 2, 5., 6.'

RESULT: The Ø array will be added or updated. It will contain the integer 2 followed by the real values 5., and 6. This is useful in some applications programs for defining table sizes in the same arrays as the tabular data.

ADDITIONAL

RESTRICTIONS: 1. The array elements cannot contain hollerith or logical information with the integer and real information.

#### 5.2.1.9 Transferring Array Elements. -

OBJECTIVE: To update or create a data base variable from an element of a data base array.

SYNTAX: 'ADD name = name (n)'

EXAMPLE: 'ADD P = J(1),  
          Q = K(2), '  
'ADD Q = K(B), ' (ILLEGAL)

RESULT: The contents of J(1) and K(2) are transferred to P and Q respectively. The element number being transferred must be a constant.

ADDITIONAL

- RESTRICTIONS:
1. The element number cannot be a data base name.
  2. The element cannot be hollerith or logical in type.

5.2.1.10 Combining Array Elements. -

OBJECTIVE: To create or update a data base variable combining data base array elements.

SYNTAX: 'ADD name = name (n) \* name (n)

EXAMPLES: 'ADD R = A \* Q(3),  
S = J(3) + Q(3)'

RESULT: The data base variables R and S will contain the combinations indicated. Any of the arithmetic operators (+, -, \*, /, \*\*) may be employed, up to 10 operations may be performed.

ADDITIONAL

- RESTRICTIONS:
1. The element number cannot be a data base name.
  2. Mixed mode arithmetic cannot be performed (i.e. integer \* real). Unpredictable results will occur.

5.2.2 The DEFINE Command. - The ADD command described in the previous paragraphs is the most useful in the communication command language. However, the DEFINE command described below will be useful for two purposes:

1. To reserve space in the data base for data bases variable before the data is actually entered.
2. To provide a brief description of the data base variables. This description is stored in the data base.

As with the ADD command, the DEFINE may be used anywhere within the execution sequence. However, it is most likely to be used in the creation of a design data base for reserving space or providing definitions for new or existing variables.

The format of the DEFINE command is:

```
'DEFINE name = n, description,'
```

The following SYNTAX or formation rules apply to the DEFINE command:

1. The opening delimiter (') may be any column.
2. No spaces may appear in the character string 'DEFINE.'
3. One or more spaces must appear between the DEFINE and the name.
4. The n is a number of data base locations to be reserved. If omitted, one will be assumed. If previously defined n will be ignored.
5. A comma must separate the name = n set and the description.
6. The description may be up to 30 characters (see Section 4 to change this number).
7. The description must be terminated with a comma (,).
8. The DEFINE command must be terminated with a DIALOG delimiter (').

More than one variable may be defined by a single define statement. Any number of continuation cards may be employed. Although more than one variable may be defined on a single card, experiments have shown unpredictable results can occur from this practice. The usual technique is to define one variable per card and use many continuation cards. The most common SYNTAX errors result from failure to conform to rules 2, 5 and 8. The reader's attention is specifically drawn to these rules.

Figure 5-7 illustrates the use of the DEFINE command. They define the example variables used in Section 5.2.1. The definitions appear on the sample data base printout on Figure 5-6.

5.2.3 The Comment Command. - The comment command is used in applications program data for identification only and as such, the comment performs no functional operation. The form of the comment:

# 7. DEFINE STATEMENT EXAMPLES \*

```

*DEFINE A, REAL VALUE          ,#
*DEFINE B, INTEGER VALUE       ,#
*DEFINE C, HOLLERITH VALUE     ,#
*DEFINE D, LOGICAL VALUE       ,#
*DEFINE E, E=A                  ,#
*DEFINE F, F=B                  ,#
*DEFINE G, G=A*2.               ,#
*DEFINE H, H=B+3                ,#
*DEFINE I, I=A+B*2.             ,#
*DEFINE J=2, REAL ARRAY        ,#
*DEFINE K=3, INTEGER ARRAY     ,#
*DEFINE L=3, HOLLERITH ARRAY   ,#
*DEFINE M=3, MIXED LOG-INT-REAL ARRAY ,#
*DEFINE N=4, SHORTHAND LOAD    ,#
*DEFINE O=3, MIXED TYPE ARRAY  ,#
*DEFINE P, P=J(1)              ,#
*DEFINE Q, Q=K(2)              ,#
*DEFINE R, R=M(3)              ,#
*DEFINE S, S=J(3)+Q(2)         ,#

```

NOTE: THE DELIMITER (#) RESULTS  
FROM THE CHARACTER SET USED  
ON THE LISTING EQUIPMENT.

FIGURE 5-7 DEFINE COMMAND EXAMPLES.

' . comment '

The SYNTAX rules for construction of a comment command are:

1. The opening delimiters and closing delimiter may appear in any column.
2. No space may appear between the delimiter (') and the dot (.) .
3. At least one space must appear between the dot (.) and the start of the comment.
4. The comment may be any number of characters.
5. The comment may appear on the same card with data.
6. The comment may not appear on the same card with another command (i.e. ADD or replacement command).
7. The comment may be continued on many cards so long as the comment does not start on the same card as data.
8. A comma (,) may not be used in a comment.
9. The comment must be terminated with a delimiter (') .

The SYNTAX rules which are most often violated are 2, 3, 6 and 8. The attention of the reader is drawn specifically to these rules.

The DIALOG executive program replaces the comment and associated delimiters with blanks as they are encountered. After processing the comment, DIALOG checks to determine if the card is all blank. If blank, DIALOG "removes" the card from the input stream.

Examples of comments are given in both figures 5-5 and 5-7.

5.2.4 Replacement Command. - Data base information is entered into the applications program input data by means of the replacement command. As the name implies the replacement command replaces delimited data base names with the corresponding values. Any delimited data base variable name may be placed on a data card of an applications program. The DIALOG executive program will replace the delimited name with the value (s) from the data base. Therefore, nearly any input procedure can be accommodated. The general format of the replacement command is:

'name'

The name is any data base name, combination of data base names and constants. Further, the names may be names of single variables or arrays. The results from the two types of replacement are different and will be treated by example. Figure 5-8 illustrates the use of the replacement command. The general rules governing the replacement are as follows:

1. The opening and closing delimiter (') may be placed in any column.
2. No space may appear between the opening delimiter and the first data base name.
3. Any combination of data base variables, array elements and constants may be used.
4. The first variable must be a name.
5. A maximum of 20 characters may be used between delimiters.
6. A maximum of 10 operations may be performed.
7. The number of significant places of the replaced number will be the maximum allowed by the space between (and including) the delimiters.
8. The opening and closing delimiter must be on the same card.

The most common SYNTAX errors result from failure to comply with rules 2, 4 and 5. Attention is specifically drawn to these rules. One additional problem may arise by not allowing enough space between delimiters to obtain the desired number of significant places.

'A'

In the above illustration, the number of significant places of the replacement value would be reduced to three places. The recommended replacement technique would be:

'A '

with sufficient space between delimiters to provide the desired significant places.

Figure 5-8 is hypothetical input stream, each line represents a card input. Delimited data base variables are placed on the card representing data base variables or combinations of data base variables, arrays and constants. The data base information comes from the data base of figure 5-6. Figure 5-9 shows the results of the replacement commands in figure 5-8. The following paragraphs discuss each of the examples with regard to objectives and results. The format of the descriptions follow the general format outlined below:

OBJECTIVE:	A concise statement of the analysis goal to be achieved by the command.
SYNTAX:	Pattern of formation of the command.
EXAMPLES:	A list of one or more examples. Sometimes incorrect examples are given and are noted as ILLEGAL. In these cases the results are either incorrect or unpredictable for the objective stated.
RESULT:	A brief description of the data base entry which will result from the use of the exemplified command.
ADDITIONAL RESTRICTIONS:	In some cases, restrictions in addition to those described above will be given.

#### 5.2.4.1 Simple Replacement of Data Base Names.-

OBJECTIVE:	To replace a data base variable on an input card with a data base value.
SYNTAX:	'name'
EXAMPLES:	'A ' 'B ' 'C ' 'D ' 'K(2)'

ARCS OF MODIFYING APPLICATIONS PROGRAM INPUT DATA BASE INFORMATION		
LINE	VALUE COMBINATION	VALUE(S)
1	A	$\neq A \neq$
2	B	$\neq B \neq$
3	C	$\neq C \neq$
4	D	$\neq D \neq$
5	E	$\neq E \neq$
6	F	$\neq F \neq$
7	G	$\neq G \neq$
8	H	$\neq H \neq$
9	I	$\neq I \neq$
10	J	$\neq J \neq$
11	K	$\neq K \neq$
12	L	$\neq L \neq$
13	M	$\neq M \neq$
14	N	$\neq N \neq$
15	O	$\neq O \neq$
16	P	$\neq P \neq$
17	Q	$\neq Q \neq$
18	R	$\neq R \neq$
19	S	$\neq S \neq$
20	T	$\neq T \neq$
21	U	$\neq U \neq$
22	V	$\neq V \neq$
23	W	$\neq W \neq$
24	X	$\neq X \neq$
25	Y	$\neq Y \neq$
26	Z	$\neq Z \neq$
27	0	$\neq 0 \neq$
28	1	$\neq 1 \neq$
29	2	$\neq 2 \neq$
30	3	$\neq 3 \neq$
31	4	$\neq 4 \neq$
32	5	$\neq 5 \neq$
33	6	$\neq 6 \neq$
34	7	$\neq 7 \neq$
35	8	$\neq 8 \neq$
36	9	$\neq 9 \neq$
37	+	$\neq + \neq$
38	-	$\neq - \neq$
39	*	$\neq * \neq$
40	/	$\neq / \neq$
41	^	$\neq ^ \neq$
42	~	$\neq \sim \neq$
43	!	$\neq ! \neq$
44	@	$\neq @ \neq$
45	#	$\neq \# \neq$
46	\$	$\neq \$ \neq$
47	%	$\neq \% \neq$
48	&	$\neq \& \neq$
49	'	$\neq ' \neq$
50	"	$\neq " \neq$
51	<	$\neq < \neq$
52	>	$\neq > \neq$
53	=	$\neq = \neq$
54	<	$\neq < \neq$
55	>	$\neq > \neq$
56	=	$\neq = \neq$
57	<	$\neq < \neq$
58	>	$\neq > \neq$
59	=	$\neq = \neq$
60	<	$\neq < \neq$
61	>	$\neq > \neq$
62	=	$\neq = \neq$
63	<	$\neq < \neq$
64	>	$\neq > \neq$
65	=	$\neq = \neq$
66	<	$\neq < \neq$
67	>	$\neq > \neq$
68	=	$\neq = \neq$
69	<	$\neq < \neq$
70	>	$\neq > \neq$
71	=	$\neq = \neq$
72	<	$\neq < \neq$
73	>	$\neq > \neq$
74	=	$\neq = \neq$
75	<	$\neq < \neq$
76	>	$\neq > \neq$
77	=	$\neq = \neq$
78	<	$\neq < \neq$
79	>	$\neq > \neq$
80	=	$\neq = \neq$
81	<	$\neq < \neq$
82	>	$\neq > \neq$
83	=	$\neq = \neq$
84	<	$\neq < \neq$
85	>	$\neq > \neq$
86	=	$\neq = \neq$
87	<	$\neq < \neq$
88	>	$\neq > \neq$
89	=	$\neq = \neq$
90	<	$\neq < \neq$
91	>	$\neq > \neq$
92	=	$\neq = \neq$
93	<	$\neq < \neq$
94	>	$\neq > \neq$
95	=	$\neq = \neq$
96	<	$\neq < \neq$
97	>	$\neq > \neq$
98	=	$\neq = \neq$
99	<	$\neq < \neq$
100	>	$\neq > \neq$

NOTE: THE DIALOG DELIMITER RESULTS FROM  
THE CHARACTER SET USED ON THE CARD  
LISTING EQUIPMENT.

FIGURE 5-8 ILLUSTRATION OF REPLACEMENT COMMAND.

# ABCS OF MODIFYING APPLICATIONS PROGRAM INPUT

## DATA BASE INFORMATION

NAME	COMBINATION	VALUE(S)
A		10.000
B		2
C		ALPHA
D		.TRUE.
E	A	10.000
F	B	2
G	A*2.	20.000
H	B+3	5
I	A+G*2.	60.00000
J		10.000000000000000000,20.0000000000000000,30.0000000000000000,
K		2, 4, 6,
L		ALPHA, BETA,
M		.TRUE., 3,
N		8.000000000000000000,8.0000000000000000,9.0000000000000000,9.0000000000000000,
O		2,5.000000000000000000,6.000000000000000000,
P	J(1)	10.0000000000000000,20.0000000000000000,30.0000000000000000,
Q	K(2)	4

FIGURE 5-9A RESULTS OF THE REPLACEMENT COMMAND.

°R	A*O(3)	60.00000
S	J(3)+O(2)	35.00000000
	N*J	80.000000000000000000,160.0000000000000000,
		270.0000000000000000,
	J*N	80.000000000000000000,160.0000000000000000,
		270.0000000000000000,
	J*2.	20.000000000000000000,40.0000000000000000,
		60.000000000000000000,
	J*A	100.000000000000000000,200.0000000000000000,
		300.000000000000000000,
	A*J	100.000000000000000000,200.0000000000000000,
		300.000000000000000000,
	A+J	20.000000000000000000,30.0000000000000000,
		40.000000000000000000,
	J+A	20.000000000000000000,30.0000000000000000,
		40.000000000000000000,
	A/J	1.000000000000000000,.500000000000000000,
		.3333333333333321491,
	J/A	1.000000000000000000,2.000000000000000000,
		3.000000000000000000,

FIGURE 5-9B RESULTS OF THE REPLACEMENT COMMAND. (CONTINUED)

RESULT: The delimited data base names will be replaced with the values in the data base as follows:

10.000

2

ALPHA

.TRUE.

The values may be real integer, hollerith or logical. The maximum number of significant places defined by the delimiter will be used. Integers are right justified in the field. All others are left justified.

ADDITIONAL

RESTRICTIONS: 1. The first element of an array cannot be used.

5.2.4.2 Simple Replacement of Data Base Combinations. -

OBJECTIVE: To replace a data base variable combination with the computed value.

SYNTAX: 'name + valuel \* name2'

EXAMPLES: 'A \* 2.'  
'B + 3'  
'A + G \* 2.'  
'A \* 0(3)'  
'J(3) \* 0(2)'

RESULTS: The delimited data base variable or array element combination will be replaced with the computed values as follows:

20.000

5

60.000

60.00000

35.00000000

The arithmetic operations are performed in a serial manner (as on a hand calculator) from the left.

ADDITIONAL

RESTRICTIONS:

1. The item adjacent to the opening delimiter must be a name.
2. Mixed mode arithmetic is not permitted (i.e. real \* integer) unpredictable results will occur.
3. The first element of an array cannot be used.

5.2.4.3 Array Replacement by Name. -

OBJECTIVE:

To replace an array name with an array of information from the data bas .

SYNTAX:

'name' or 'name(1)'

EXAMPLES:

'J' or 'J(1)'

'K'

'L'

'M'

The arrays are placed on the card starting at the first delimiter using 20 characters per element, three elements per card. Elements are separated by commas. Continuation cards are created as required with data starting in column 2. The position of the closing delimiter is immaterial. 'J' or 'J(1)' produce the same result. Integers are right justified in the field, all others are left justified.

ADDITIONAL

RESTRICTIONS:

1. Array replacement is generally limited to NAMELIST or special input procedures. It is probably not suitable for formatted input.

#### 5.2.4.4 Array Replacement of Data Base Combinations. -

OBJECTIVE: To replace a data base combination on an element by element basis.

SYNTAX: 'name1 \* name2'

EXAMPLES: 'N\*J' (array \* array)  
'J\*2.' (array \* constant)  
'J\*A' (array \* data base variable)  
'A/J' (constant/array)

RESULTS: The operation is performed on an element by element basis. If one factor is a data base variable or constant, the one number is used as an operator on every element of the array. If both are arrays, the resulting array is equal in length to the shorter one. Division of a constant by an array results in an element by element division of the constant by the elements of the array. For multiple operations, an element by element serial arithmetic is performed.

ADDITIONAL  
RESTRICTIONS: Reference to the first element of an array refers to the entire array.

A summary of the communication commands is given in Appendix B.

### 5.3 Standard Utility Procedures

The purpose of maintaining a program library for use with the DIALOG executive system is primarily for ready availability of applications programs. However, during the development and subsequent applications of the system, a set of utility programs or procedures has evolved for performing such tasks as:

1. Compilation and execution of interface programs.
2. Disposition of data files.
3. Report writing.

#### 4. Executing arbitrary control card procedures.

These procedures are stored in the control card data base in exactly the same manner as applications program procedures. They are called into execution with the EXECUTE directive. The following paragraphs provide a brief description of the use of the standard utility procedures. Some have associated data and some do not.

5.3.1 COMPL1/MYPG1: Compile a FORTRAN Program/Execute the Compiled Program. - COMPL1 compiles the program while MYPG1 executes the compiled program. The data associated with COMPL1 is the FORTRAN source code. The data associated with MYPG1 is the input data for the compiled program. The execution sequence is as follows:

```
'EXECUTE COMPL1'
```

```
FORTRAN Source Code
```

```
$EF$
```

```
'EXECUTE MYPG1'
```

```
Data for Compiled Program
```

```
$EF$
```

5.3.2 REPORT: Generates Data Base Status Report. - Usually the submission of a computation to the digital computer results in the generation of detailed information about the process involved. The results as well as intermediate information are printed by the normal output channels. The submission of the same computation using the DIALOG executive system involves the generation of the same information plus some summary type information which is usually a small subset of the total output of the applications program. The summary information is placed on the special output file discussed in Section 4.

The information on the special output file may be placed in the data base after which it is generally available for printing through the REPORT procedure.

```
'EXECUTE REPORT'
```

```
report data
```

```
$EF$
```

The report data is a sequence of punched cards much like the input data to an applications program. However, the report data is not processed by any computer program but simply printed by the DIALOG executive program.

The report data is formatted by the analyst to provide any descriptive information desired. Further the report data may contain data base information through the use of the communication commands described in Section 5.2. An example card in the report might be:

WEIGHT OF THE SYSTEM IS 'WGT' POUNDS

In the above illustration, WGT is a data base variable. The DIALOG executive program replaces the data base name and delimiter 'WGT' with the information stored in the data base. The report is printed after processing the report data. The result is a "stylized report" specifically tailored to the needs of the analyst. The report may contain "carriage control characters" in column 1 of the report data cards.

1 - eject a page before printing

0 - skip a line before printing

Any number of reports may be generated during a simulation. Usually during the initial phase of coordinating of large simulation using the DIALOG executive system, the staff selects subsets of the data base information to be communicated to each staff member for analysis. The format of the individual reports is tailored to the needs of the individual receiving the information. Once the format is established, it is keypunched on data cards with data base information being identified by name in the manner described in Section 5.2. These data cards become a report file.

A mini-report exemplifying this technique is shown in figure 5-10. Any of the features of the DIALOG language including scaling and adding data base information are used in a completely free field report format. The first column of each card is reserved for printer carriage control providing a convenient means of paging and spacing for report clarity. Figure 5-10 also shows the printed results of the report file with data from the data base.

EXECUTE REPORT #  
 4. DATA FOR SUMMARY REPORT #  
 PAGE 1

SUMMARY REPORT FOR ODIN/RLV  
 ELAPSED TIME = #ELTIME# CPU SECONDS  
 1 CYCLE(S)  
 ADD #ATCOST# #TCOST#  
 ADD #N#1.689#  
 ADD #N#0.45359#  
 MAXIMUM PAYLOAD ..... #PAYLO# # LBS #PAYLO#K# KILOGRAMS  
 BOOSTER HEIGHT ..... #GROSS# # LBS #GROSS#K# KILOGRAMS  
 ORBITER HEIGHT ..... #GROSS# # LBS #GROSS#K# KILOGRAMS  
 TOTAL COST ..... #A# # MILLIONS  
 BOOSTER COST ..... #ATCOST# # MILLIONS  
 ORBITER COST ..... #ATCOST# # MILLIONS  
 STAGING VELOCITY ..... #VSTG# # FPS #VSTG#K# KNOTS  
 ENGINE VACUUM THRUST ..... #THRUST# # LBS  
 BOOSTER MASS RATIO ..... #ALPHA(2)#  
 ADD #N#JUPES#  
 BEST CYCLE=#JUPES#, THE #N#2-17TH RECORD ON FILE OUTSAV. N=#N#  
 END OF ODIN/RLV SUMMARY  
 REPORT  
 00000000000000000000

PAGE 1

SUMMARY REPORT FOR ODIN/RLV  
 1 CYCLE(S) ELAPSED TIME = 58.26500 CPU SECONDS  
 MAXIMUM PAYLOAD ..... 31495.5675 LBS 14444.0500 KILOGRAMS  
 BOOSTER HEIGHT ..... 267117.97 LBS 1211592.30 KILOGRAMS  
 ORBITER HEIGHT ..... 71746.521 LBS 323494.319 KILOGRAMS  
 TOTAL COST ..... 357499.5935 MILLIONS  
 BOOSTER COST ..... 24245.2325 MILLIONS  
 ORBITER COST ..... 518974.3540 MILLIONS  
 STAGING VELOCITY ..... 9212.29150 FPS 5448.36683 KNOTS  
 ENGINE VACUUM THRUST ..... 525200.000 LBS  
 BOOSTER MASS RATIO ..... 2.7250000  
 BEST CYCLE= 1. THE 110 RECORD ON FILE OUTSAV. N= 1  
 END OF ODIN/RLV SUMMARY REPORT

FIGURE 5-10 ILLUSTRATION OF REPORT GENERATION CAPABILITY.

5.3.3 ENDODN: To Save a Design Data Base for Future Use. -  
Usually the execution of the DIALOG executive system requires the creation of a design data base which is used and modified during the simulation process but not saved at the end of the run. It is sometimes desirable to save the design data base and restart the simulation at a later date. Examples of this requirement would be a very long simulation involving many program executions or an optimization involving repetitive evaluations through a series of application programs.

The ENDODN procedure allows the analyst to save the data base in the precise configuration required to restart the simulation. The control card sequence follows:

```
ENDODN =  
  
      @ASG,CP  DATAB.,F  
  
      @COPY    DBASE.,DATAB  
  
      @FIN
```

It is not essential (nor required) to execute this procedure using the EXECUTE directive. If the ENDODN procedure is stored in the control card data base, it will be executed at the proper time (after the END directive). The data cell location must be specified by the analyst and retrieval as part of the initialization procedure. The retrieval is discussed in Section 5.1.2.

#### 5.4 Special Options in DIALOG Executive Program

Generally any name, except communication command names, may be used to identify data base information. However, there are a few additional exceptions. The excluded names are:

```
BUILD  
DBDUMP  
ELTIME  
INDUMP  
OUTDMP  
PAGDMP
```

which represent special commands to DIALOG. The data base variables are stored with the ADD command (i.e. 'ADD BUILD').

If present in the data base, DIALOG will perform special functions as described in the following paragraphs.

The BUILD option is associated with the dynamic construction of the design data base by the applications programs. Two values have significance to the DIALOG executive program.

BUILD = 0

BUILD = 1

The zero value specifies that previously undefined variables will not be defined by an applications program. A value of one specifies that all information from the previous program will be stored in the data base regardless of its previous data base status. The value of BUILD may be changed from program-to-program by use of the ADD command:

'ADD BUILD = n'

in the input data of the applications program. The change in the BUILD option becomes effective for the program where it occurs and remains effective until changed again.

The BUILD option only affects data coming from applications programs. It has not affect in ADD commands or other user communication commands.

DBDUMP is a DIALOG option which specifies that the entire data base be printed after each applications program execution. An example of this printed output is shown in figure 3-11. It should be noted that this option is mandatory when selected. Selective printing of the data base is accomplished by the PRINT directive.

'PRINT DBASE'

The directive illustrated above is discussed in Section 3.1.

ELTIME is a timing option. It is selected by a setting ELTIME to an initial value:

'DEFINE ELTIME = 7, description,'

When selected, a timing routine in DIALOG will be activated. This routine monitors the individual and cumulative computer

processing parameters for the applications programs. The parameters are identified on the printed output.

INDUMP is an optional data base name which specifies the printing of the modified input stream for the applications programs. The modified input stream represents the input file in exactly the form the applications program will read it. The INDUMP option is useful in the early phases of linking programs for debugging purposes.

OUTDMP is an optional data base name which specifies the printing of the special data base output file, NMLIST, which contains all the information available for entry into the data base. Details of the NMLIST file and how it is used are discussed in Section 3.2.4. The OUTDMP can be used to determine what information is available from a given applications program or simply as a debugging aid in the early phases of the analysis.

PAGDMP is an option available to the programmer for detecting errors within the DIALOG executive system. It has little use to the analyst using the system. It is mentioned only because PAGDMP is an excluded data base name which has special significance to the DIALOG executive system.

A summary of special options and additional restrictions is given in Appendix C.

## 6.0 APPLICATIONS

The DIALOG executive system was developed to provide the program linking capability required in the Optimal Design Integration (ODIN) procedure of Reference 9. Two applications were used to demonstrate the ODIN capability and are presented here to illustrate the use of the DIALOG executive system in linking independent programs. The ODIN application program library is shown in figure 6-1. This library was used to perform the analysis presented in this section. The results indicate that the DIALOG executive system has unlimited potential in solving a wide variety of engineering problems. The examples presented were performed on the CDC 6600 at Langley Research Center.

### 6.1 Orbiter Landing Skin Temperature Study

An example of a small ODIN problem is shown in figure 6-2. A study was required to determine if landing performance or stability and control might be affected by the presence of excessive skin temperatures on the Space Shuttle Orbiter. This problem was formulated in ODIN to determine skin temperature time histories using the various ODIN technology programs. The MINIVER (a mini-version of the MDAC JATO Aerodynamic Heating Program) and ABLATOR (reference 11) programs required were quickly integrated in the ODIN system. MINIVER was used to obtain convective heat rates along the entry trajectory and ABLATOR enabled calculations of the associated skin temperature variations over the orbiter surface. Formulation of the problem required the combined efforts of a group of engineers with technological backgrounds in materials, flight dynamics and aerothermodynamics. The deck setup for the problem is shown in figure 6-3. The problem was solved approximately one week after its conception. The results, shown in figure 6-4, indicated that no excessive temperatures were present on the orbiter skin during approach and landing using either reusable or ablative material.

### 6.2 Shuttle Orbiter Wing Design Study

A somewhat more complex demonstration of the ODIN system is summarized in figure 6-5 for a shuttle orbiter wing design study. The purpose of this study was to provide an orbiter wing which would achieve hypersonic/subsonic compatibility. Study guide lines were chosen in accordance with those from NASA's Shuttle Request for Proposal in

## ODIN APPLICATIONS PROGRAMS

### AERODYNAMICS

SKINF  
DATCOM  
HABACP  
TREND  
VDRAG

### GEOMETRY

CONFIGURATION  
WETTED  
PANEL

### TRAJECTORY

ATOP  
PRESTO  
POST

### WEIGHTS

WAATS  
VAMP

### STRUCTURES

SSAM  
AFSP

### HEATING

MINIVER  
ABLATOR

### OPTIMIZATION

AESOP

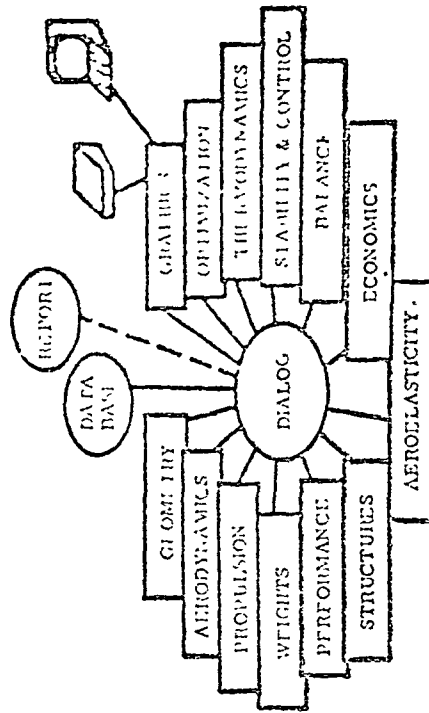


FIGURE 6-1 ODIN APPLICATIONS PROGRAMS

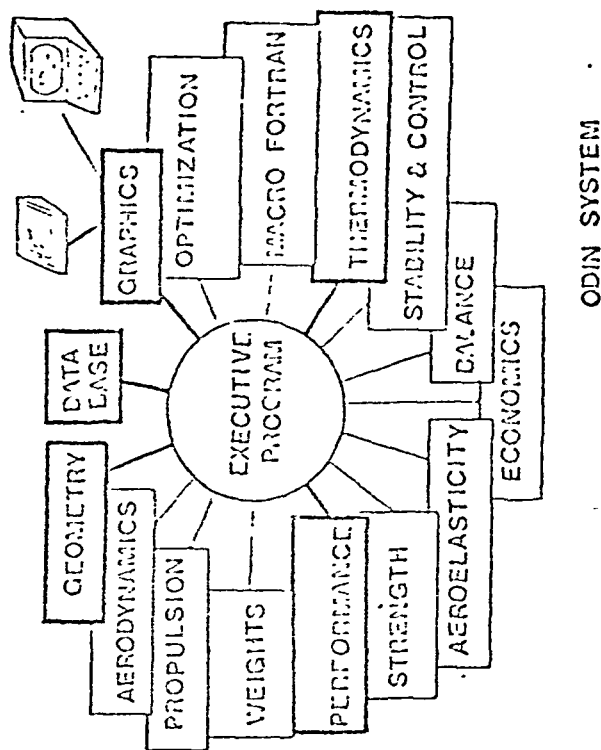
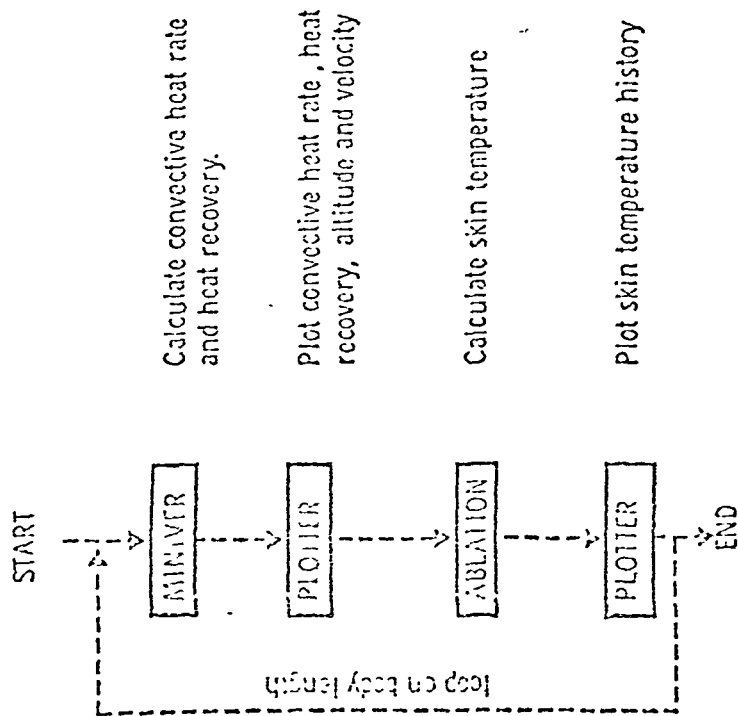


FIGURE 6-2 LANDING SKIN TEMPERATURE STUDY FOR ORBITER

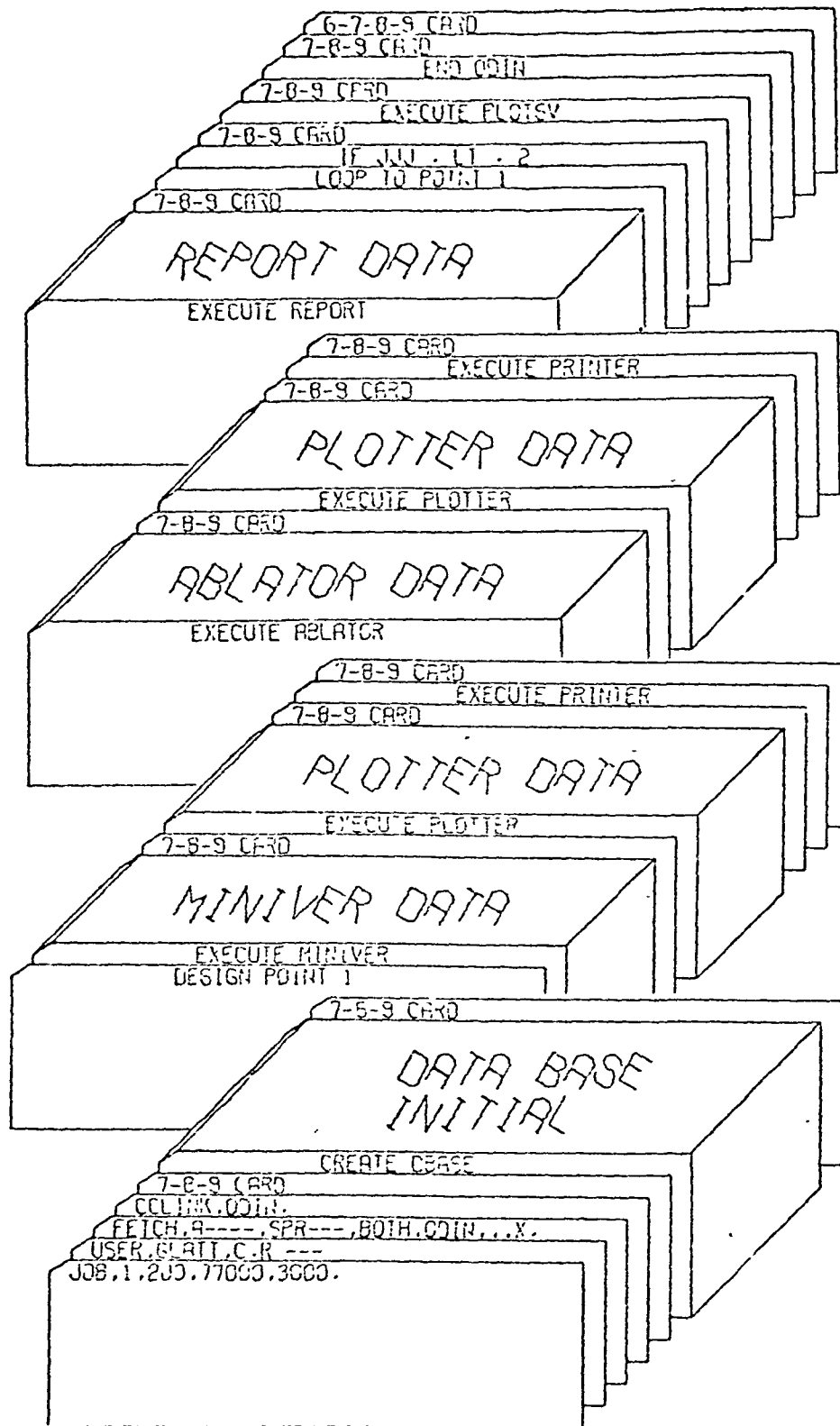
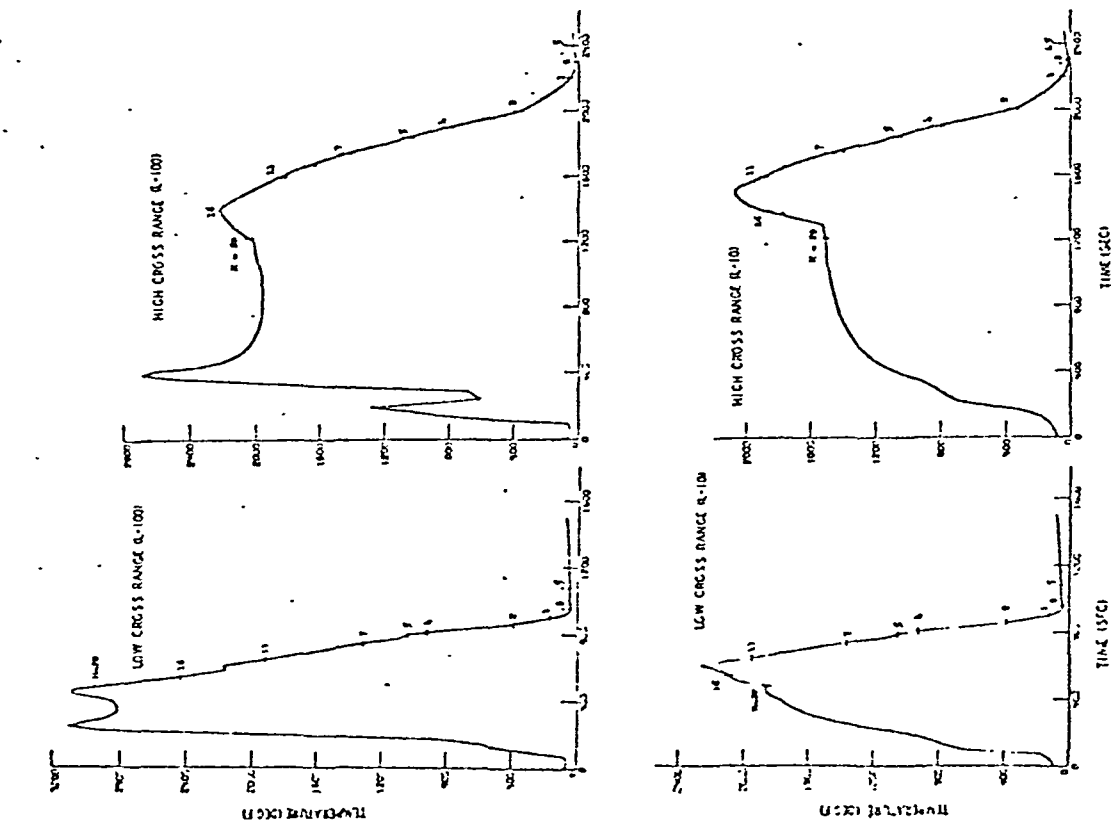


FIGURE 6-3 DECK SET UP FOR ORBITOR LANDING SKIN  
TEMPERATURE STUDY

# SKIN TEMPERATURE HISTORY 3-INCH MARTIN SLA-561 ABLATOR



# SKIN TEMPERATURE HISTORY RSI MATERIAL

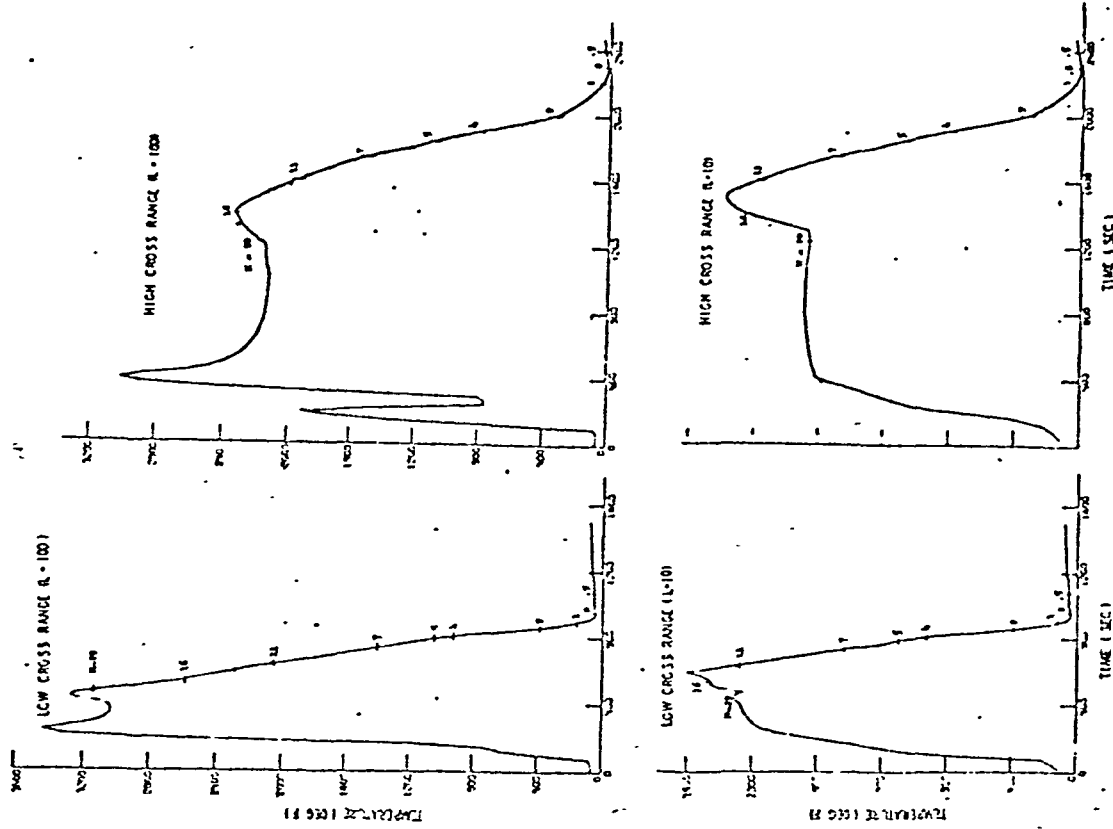


FIGURE 6-4 RESULTS OF THE ORBITER LANDING SKIN TEMPERATURE STUDY



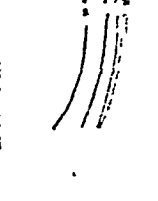
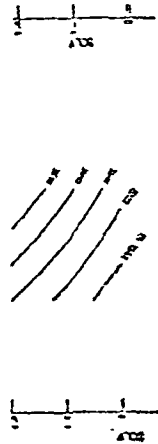
February, 1972. The orbiter wing geometry was perturbed over a matrix which yielded design data for 37 possible configurations. The ODIN design sequence progressed downward in the figure for each wing design and ended with aerodynamic data plots, a configuration drawing and a summary report of geometric and aerodynamic performance data. The technique allowed the users to converge rapidly on a viable orbiter wing design which was subsequently proven with the aid of minimal wind tunnel development. Figure 6-6 illustrates the type of results from the orbiter wing design study.

LANGLEY RESEARCH CENTER ODM/RLV GEOMETRY PROGRAM  
 54114-00000 54114-00000 XOF-49.53

ORBITER CHARACTERISTICS  
 (Specify Wing Area of 51 ft. or 100 ft.)

P 100 ft. Wing Area Factor in 2-direction  
 P 100 ft. Wing Area Factor in 1-direction  
 PITCH (AROUND) ANGLE OF ATTACK  
 LENGTH - 40 ft

LANDING HEIGHT (XOF-49.53)



THEORETICAL ASPECT RATIO (XOF-49.53)



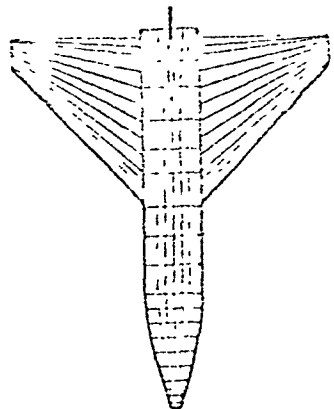
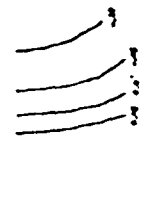
LANDING SPEED (XOF-49.53)



THEORETICAL AREA (XOF-49.53)



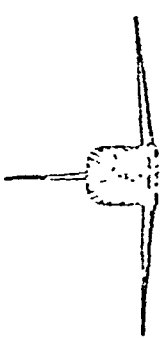
STATIC MARGIN (XOF-49.53)



TOP VIEW



SIDE VIEW



FRONT VIEW

FIGURE 6-6 TYPICAL RESULTS FROM THE ORBITER PITCH TRIM MISMATCH STUDY

## 7.0 CONCLUSIONS

A very large scale synthesizing system for engineering processes has been described. The elements of the system are a library of independent applications computer programs, an executive computer program and a data base which forms the common information link among the independent applications programs. The programs can be used by individuals for small problems or the operation can employ the design team approach. In the latter case, the design team defines the program sequences, data interfaces and matching loops required to achieve the desired design objective. The system provides the users with the ability to formulate the computer aided design problem at the task level in much the same manner as is employed in the industrial design process.

The executive computer program DIALOG controls the sequence of execution of the independent program elements and performs the data management function through the maintenance of the data base of common information. Each program is executed sequentially and as such is "unaware" of its contribution to a larger engineering process. DIALOG interrogates the data into and out of the independent programs and performs data manipulations according to "instructions" embedded in the data. The "instructions" are user supplied and form the control and communication language which are input to the DIALOG executive program. DIALOG restructures the input stream based on the instructions. The result is a flow of information which is not unlike the normal flow of individual jobs.

The greatest single advantage of the DIALOG executive system is that it allows full use of virtually all past developments in engineering technology for the synthesis of engineering processes. Any existing checked out computer code can be easily incorporated into the system library and the developer of new technological modules is unconstrained by requirements of the DIALOG executive system. Little or no programming knowledge is required to incorporate a program into the system for the first time.

The control and communication language consists of a simple and easily understood set of instructions which provide the capability of creating a network of computer programs for analysis at any level of detail. All synthesis processes and data intercommunication are performed at the program

input level. Conditional branching logic is provided for creating sizing and/or optimization loops within the synthesis. There is no effective limit to either the number of programs used or the complexity of design loops created.

The manual data transfer from technology to technology may be drastically reduced using the DIALOG executive system. Further, the chance of data error, data misunderstanding or data misrepresentation is virtually eliminated. All factors dealing with "engineering judgment," "design margins" and "non-optimum analysis" may be employed and are visible to the design team.

Data visibility has been a key requirement in the development of the DIALOG executive system. Report generation is an integral part of DIALOG. User generated reports based on data base information can be generated at any point in the sequence of program executions. A variety of graphical capability is available in the program library.

Finally the DIALOG executive system provides a true building block approach to the synthesis of engineering processes. Applications programs may be added, deleted or replaced to suit the design objective. This provides a responsiveness of computer aided design techniques never before available to the designer. All or any part of the design process may be synthesized but when using the DIALOG executive system, the designer never relinquishes his option to perform the analysis by alternate means, including hand calculation.

The software associated with the DIALOG executive system is written in the FORTRAN source language and is relatively machine independent. Machine dependent and system dependent code is used only when absolutely essential to the proper function of the DIALOG executive system. Where used the machine dependent code is isolated for quick conversion to other machines and other systems. Versions of the DIALOG executive system for CDC 6000 series and Univac 1100 series computers have been developed.

## 8.0 REFERENCES

1. Gregory, T.J., Peterson, R.J. and Wyss, J.A.: Performance Trade-Offs and Research Problems for Hypersonic Transports. AIAA Journal of Aircraft, July-August 1965.
2. Peterson, R.H., Gregory, T.J. and Smith C.L.: Some Comparisons of Turboramjet-Powered Hypersonic Aircraft for Cruise and Boost Missions. Journal of Aircraft, September-October 1966.
3. Gold, R. and Ross, S.: Automated Mission Analysis Using a Parametric Sensitivity Executive Program. AAS Paper 68-146, presented at the AAS/AIAA Astronautics Specialist Conference, September 1968.
4. Wennegal, G.J., Mason, P.W. and Rosenbaum, J.D.: IDEAS: Integrated Design and Analysis System. SAE Paper 68-0728, presented to SAE Aeronautics and Space Engineering Meeting, October 1968.
5. Adams, J.D.: Vehicle Synthesis of High Speed Aircraft, VSAC, Volume I, USAF AFFDL-TR-71-40, 1971.
6. Oman, B.: Vehicle Synthesis for High Speed Aircraft, VSAC, Volume II. USAF AFFDL-TR-71-40, 1971.
7. Lee, V.A., Ball, H.G., Wadsworth, E.A., Moran, W.J. and McLead, J.D.: Computerized Aircraft Synthesis. AIAA Journal of Aircraft, September-October 1967.
8. Herbst, W.B. and Ross, H.: Application of Computer Aided Design Programs for the Management of Fighter Development Projects. AIAA Paper 70-364, presented at the AIAA Fighter Aircraft Conference, March 1970.
9. Hague, D.S. and Glatt, C.R.: Optimal Design Integration of Military Flight Vehicles - ODIN/MFV. AFFDL-TR-73-123, 1973.
10. Morris, Robert: Scatter Storage Techniques, Communications of the ACM. Volume II, No. I. January 1968.
11. Swann, R.T., et al: One-Dimensional Numerical Analysis of the Transient Response of Thermal Protection Systems. NASA TN-D-2976, 1965.

## APPENDIX A    CONTROL DIRECTIVE SUMMARY

- \* 'EXECUTE name'            A directive for executing a sequence of control cards by name. Any name for which a prestored set of control cards has been defined is legal.
  
- \* 'INITAL name'            File handling directive for initializing files; the two acceptable names are:
  - AESOP - parameter optimization file
  - DBASE - design data base
  - CCDATA - control card data base
  
- \* 'UPDATE name'            File handling directive for updating files. The two acceptable names are:
  - DBASE - design data base
  - CCDATA - control card data base
  
- 'DESIGN name'            Control directive defining a point in the execution sequence for which control may be returned. The name cannot be the same as a data base variable.
  
- 'LOOP TO name'            Branching instruction referring to a design name. It can be conditional or unconditional.
  
- 'IF name.OP. name'        Condition for branching. Any number of conditions may be specified on separate cards after a LOOP directive. If more than one condition is specified, the logical .OR. is implied. That is, any one of the conditions satisfied will trigger the branch instruction.
  
- 'RESTRT'                  Means use the existing data base. It must have previously been defined and stored.
  
- 'PRINT name'              File handling directive for printing files DBASE and CCDATE are optional names.

OP is a conditional operator (LT,LE,EQ,GE,GT)

- \* Data is expected; end of record (789) is required.

## APPENDIX B    COMMUNICATION COMMAND SUMMARY

'ADD A = B, ...'

Used to create a new data base entry or alter the information associated with an existing data base entry.

A is a new or existing data base entry, scalar or vector.

B is the update information which can be real, integer or logical constants, variables or scaled combinations of scalar or vector elements.

Multiple commands can be executed with a single ADD statement.

'DEFINE A = n, description,'

Used to define new or existing entries in the data base.

A is the new or existing data base entry.

n is the desired number of data base locations. It is ignored if the entry exists, the default is 1 if omitted.

description is the hollerith information associated with the variable A.

' .comment'

Used for placing descriptive information in the data stream. DIALOG replaces the comment and associate delimiters with blanks in the applications program data deck.

'A

'

Used to replace data base names and delimiters on an input card with data base information.

A may be a scaler or vector data base entry of real, integer, hollerith or logical type.

A may be a combination of real or integer data base variables, array elements or constants.

## APPENDIX C EXCLUDED NAMES FOR DATA BASE VARIABLES

Generally, the user of the DIALOG Executive System is free to specify names for data base variables and arrays. However there are certain names which are excluded. The excluded names are those used by the DIALOG executive system for option specification and by the analyst for 'DESIGN identifiers.' Certain other miscellaneous names are excluded.

### DIALOG Executive System Options

BUILD	Option for dynamic construction of the data base.
DIVIDE	Symbol used for divide (/) in the operator directory.
DOLLAR	Symbol used for DOLLAR (\$) in the operator directory.
ELTIME	Total elapsed simulation time used in timing option.
EQUAL	Symbol used for equal (=) in the operator direction.
EXPON	Symbol used for exponentiation (**) in the operator directory.
MINUS	Symbol used for subtraction (-) in the operator directory.
MLTPLY	Symbol used for multiplication (*) in the operator directory.
NOTEQL	Symbol used for a data base delimiter (&) in the operator directory.
INDUMP	Option to print the modified input for every program
OUTDUMP	Option to print the special data base output file from each program.
PAGDMP	Option to print the internal string processing information
PLUS	Symbol used for addition (+) in the operator directory.

## DESIGN Identifiers

The specification of a DESIGN identifier by the control directive:

'DESIGN identifier'

results in the identifier being stored in the data base. Once used the identifier is excluded from use as a data base variable name.

## Miscellaneous Exclusions

Generally, the names listed above are the only exclusions the analyst need be concerned with. However, it is recommended that the communication command names and control directive names be excluded also.

ADD	Add command.
DELETE	Delete command.
DEFINE	Define command.
.	comment 'command'
INITIAL	Create directive.
DESIGN	Design directive.
EXECUTE	Execute directive.
IF	If directive.
LOOP TO	Loop To directive.
RESTART	Restart directive.
UPDATE	Update directive.

## APPENDIX D - DIALOG EXEC II SYSTEM

### Introduction

A version of the DIALOG executive system is tailored to the EXEC II operating system on the Univac 1108. The applications programs comprising the system were installed and tested with two ODIN test problems.

Two new programs were developed for the storage of "catalogued procedures" and the storage of simulation data. Catalogued procedures are standard control card sequences and are stored in the "PCF" areas. The procedures are executed with the ADD software now available on the EXEC II system.

Simulation data which usually is quite cumbersome for large simulations can be stored on magnetic tape and recalled later for the performance of simulations. Further, the data can be updated by card with a procedure similar to that available for updating FORTRAN programs stored in the PCF.

## THE DIALOG EXECUTIVE SYSTEM

The DIALOG executive system is uniquely structured to perform a data linkage between independent applications programs sequentially executed in the same computer run. The Univac 1108, EXEC II version developed specifically for the JSC computer complex, is similar to the currently operational EXEC VIII described in this report. However, hardware and software differences dictate some departure from that system. Figure 1 shows the structure of the EXEC II system at JSC.

The following paragraphs describe the system (as currently configured) and its use. In figure 1 the data flow is downward, each block representing a major control function.

In operation the CUR utility is used to load the ODIN library into the PCF from a magnetic "simulation tape" file. Although not essential to the operation of ODIN, the recommended procedure is to store the programs in absolute element form for highspeed loading. The use of a simulation tape (SIMTAP) accomplishes two objectives:

1. Reduces the number of tapes required to run an ODIN simulation.
2. Reduces the number of PCF elements in the PCF area.

The source and relocable binary files for the ODIN programs are maintained separately on other tape files and used as required to create or update the SIMTAP. The SIMTAP approach has been used before but is usually not required where on-line permanent file storage is available. The actual creation of a SIMTAP will be discussed in a later section. More than one SIMTAP may be required and some users of ODIN may elect to maintain his own SIMTAP (s) even though some programs on his tape may duplicate existing programs residing elsewhere.

After the ODIN library is loaded the MAING program is executed. MAING performs a data editing function with the ODIN input data. The input usually stored on tape is read by MAING and transferred to the logical unit (22) used by DIALOG for reading input data. The ODIN user has the option of modifying the input data in a manner similar to that for FORTRAN programs stored in the PCF. The procedure for performing updates as well as initial storage of the data is described later.

Referring again to figure 1, the DIALOG executive system is initialized by the program DBINIT. The primary functions are the initialization of the data bases.

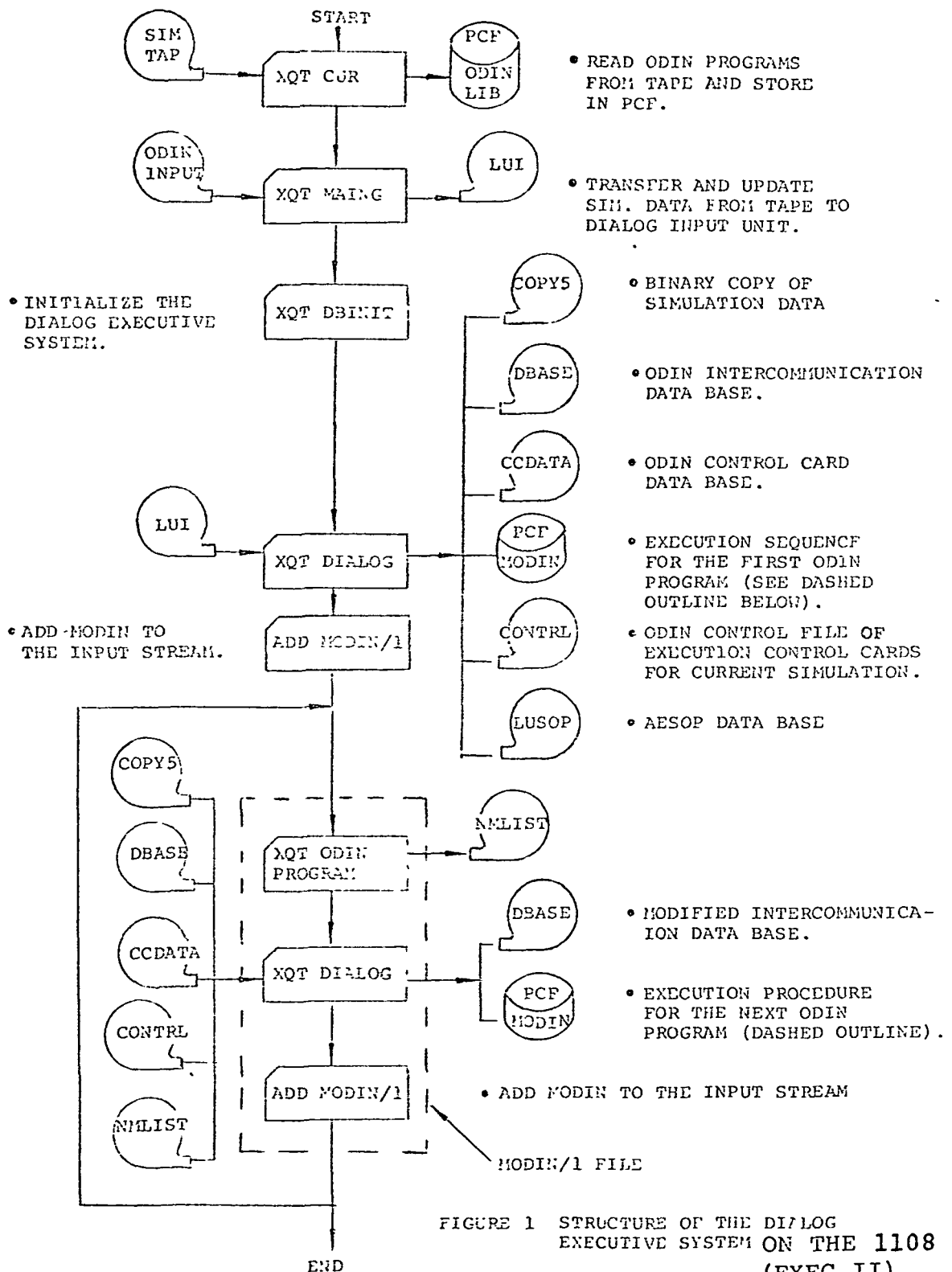


FIGURE 1 STRUCTURE OF THE DIALOG EXECUTIVE SYSTEM ON THE 1108 (EXEC II)

On the initial DIALOG execution the simulation data is read from LUI, the DIALOG input file. Based upon the instructions contained in LUI, DIALOG constructs several files. COPY5 is a binary (fast read) copy of the simulation data which will be used during the simulation to construct input files for the simulation programs. DBASE is an unstructured file of name oriented data which is initially constructed by DIALOG and maintained throughout the simulation. CCDATA is similar in construction to DBASE but contains control card data for the execution of ODIN programs. CCDATA is used by DIALOG in the construction of input files. It may be constructed during the simulation or saved from simulation to simulation. The CONTROL file is a list of control cards for the execution of the program sequence. This file is constructed during the initial pass through DIALOG and used (in conjunction with COPY5) to construct input files for simulation programs. MODIN contains the input data for the next program or sequence of programs. This file is constructed each time DIALOG is executed and contains the following information:

- control card (s) for next program
- data for next program
- control card (s) for the reexecution of DIALOG

An example of a MODIN file is shown in figure 2. MODIN is stored in the PCF by DIALOG so it will be available to be "added" to the normal input stream. LUSOP is the AESOP data base. This file is initialized on user command whenever the parameter optimization program, AESOP is used in a simulation. A separate LUSOP file must be maintained for each AESOP loop created in the simulation.

After the initial pass through DIALOG the MODIN file for the first program will have been constructed. The next control card (shown in figure 1) is an ADD card. This control card adds the contents of the MODIN file to the input stream such that the operating system immediately begins executing from that file. Upon processing of the contents of MODIN, the usual procedure is to return to the control cards on the input file. However, the unique feature of the MODIN file is that it contains a control card for the reexecution of DIALOG, which constructs a new MODIN file, and an ADD card for the new file which DIALOG just built. This scheme is illustrated in the lower portion of figure 1 by the dash outlined region representing MODIN. The simulation sequence is 'bootstrapped' along by DIALOG based on instructions contained in the simulation data.

3 ELT MODIN/1.1.730331, 67513

000001	LN XQT ROZEN,ROZEN	Control card for next program execution
000002	SRBROKI	
000003	X=-2.0000000,	Input data for next program.
000004	Y=-2.0000000,	
000005	S	
000006	@ XQT OCLOG	Control cards for the re-execution of DIALOG.
000007	@ XQT CUR	
000008	-LIST MODIN/1	
000009	@ ADD MODIN/1	Control card for adding the next MODIN file to input.
END CUR LCC 1102-0396 L9		

FIGURE 2 ILLUSTRATION OF THE MODIFIED INPUT FILE.

## USE OF DIALOG EXEC II VERSION

The DIALOG executive system is user oriented and requires little more knowledge to use than to use the individual programs involved. The user simply sets up the input decks for each simulation program which he intends to use, places the interface data into the input stream and replaces the normal control cards with DIALOG control directives. The objective in using DIALOG is to link the independent programs and communicate information among them. There are a total of twelve instructions available to perform the linkage. These instructions are summarized in Appendixes A and B but a more detailed description is presented in Section 5. This section is presented to aid the user at JSC in setting-up a simulation on the Univac 1108 (EXEC II) computer. Some knowledge of the control and communication language is a prerequisite to actually using the DIALOG executive system.

Usually the submission of a simulation requires control cards similar to those shown in figure 3. The assign cards shown are for the simulation tape (SIMTAP) and the data tape (DATAPE) containing the stored input data for the simulation. The creation of a SIMTAP will be discussed below. An example of the contents of a data tape is shown in figure 4. Data tapes may be created within an ODIN simulation as described below. The CUR operation (figure 3) loads all ODIN programs into the PCF from the simulation tape.

The first ADD card (INPUT/DATA) prepares the ODIN input data for use in the simulation. The second ADD card (ADD ODIN/XQT) transfers control to the DIALOG executive system. All further computer operations are controlled by DIALOG based upon the instructions encountered in the simulation data.

### Manipulation of ODIN Input

The ADD INPUT/DATA control card (figure 3) executes the MAING program which permits modification to the ODIN input data. For example, suppose the user wished to alter the equation for computing F (card 25 - figure 4) and to add a data base print command after the report (card 46). The modifications to the ODIN input on C (figure 4) would be as follows:

```

7   ASG B = _____ ( Simulation Tape )
8
7   ASG C = _____ ( Data Tape )
8
7   XQT CUR
8   TRW B
   IN B
   TRI B
* 7   ADD INPUT/DATA
8   ( MODIFICATIONS TO
   ( ODIN INPUT ON C )
* 7   ADD ODIN/XQT
8

```

\* NOTE: Only one space is permitted between the master space and the ADD.

DDM11106 EXEC 11 VERSION 1.0 APRIL 17, 1973		DATE J10373
INITIAL DEASE6		1
6ADD ALPHA=-3+,-3+		2
F=C,		3
JJJ=0,		4
X=3+ Y=3+		5
6		6
SEFS		7
6INITIAL CCDATA6		8
6FORTRAN		9
6 FOR DDINPI		10
NEEPG=		11
ON XOT DDINPI		12
DIALOG		13
6 XOT DDLOG		14
6 XOT CUR		15
LIST HODIN/1		16
6 ADD PDDIN/1		17
ENDCON=		18
6 EOF		19
6		20
SEFS		21
6EXECUTE FORTRAN		22
NAMELIST /10/X,Y,F		23
READ (5,10)		24
F=J**2-Y**2		25
WRITE (14,10)		26
END		27
SEFS		28
6DESIGN-POINT16		29
6PRINT DEASE6		30
6EXECUTE NEEPG=		31
6ADD JJJ=JJJ+16		32
610		33
X=6X 6 Y=6Y 6		34
SEND		35
SEFS		36
6EXECUTE REPORTS		37
1 DIALOG HIDE EXEC 2 TEST CYCLE 6JJJ6		38
0 COMPILE AND EXECUTE FORTRAN PROGRAM		39
0		40
0 X=6X 6		41
Y=6Y 6		42
0 F=6F 6		43
6ADD X=X+1, Y=Y+1, 6		44
1		45
SEFS		46
6LOOP TO POINT16		47
6IF JJJ.LT.3 6		48
SEFS		49
6END DDIN6		50
ENDEND		51

FIGURE 4 ILLUSTRATION OF A DATA TAPE

```

-25,25
      F = X**2+2.*Y**2
-46
      &PRINT DBASE&
-ENDEND

```

If no modification to ODIN input is required, the user need not put any data in after the ADD INPUT/DATA control card. The data will simply be transferred as is to the DIALOG input file. If the user wishes to read in all data, the data tape need not be assigned. The following procedure would apply:

```

78 ADD INPUT/DATA

-UNITS  }
5,22    }      • logical units for input will be read
              (INPUT = 5, OUTPUT = 22)

      ( ODIN input
        data )

-ENDEND      • end of DATA

```

In the above illustration, the input unit is 5 (or cards). The output unit is 22 (the unit on which DIALOG expects the data).

If the user wishes to store the data for the first time on a data tape before the simulation starts, the data tape (C) would be assigned and the following procedure would be employed:

```

78 ADD INPUT/DATA

-UNITS  }
5,3     }      • Transfers new data to Tape C
              ( new data )

-ENDEND      • Transfer data from Tape C to DIALOG
              INPUT UNIT (22).

```

The data would then be ready for use by DIALOG.

In summary, each time the control card:

#### 7<sub>8</sub> ADD INPUT/DATA

is used, data will be transferred from one unit to another with possible user modification. The input and output units may be specified by the user. If not specified, units 3 and 22 respectively will be used by the program.

Two data tapes have been created at JSC for use in testing and demonstrating the ODIN system. These are tabulated in Table I.

#### Manipulation of ODIN Programs

The established procedure for using ODIN on the Univac 1108 (EXEC II) system requires the use of a simulation tape (SIMTAP). The SIMTAP must be prepared before simulation may begin. As a minimum, SIMTAP must contain the following programs:

DBINIT

DIALOG

MAING

Any other user programs called for in the simulation must also be stored on SIMTAP. Usually the ODIN programs are stored in absolute element form for fast loading. This is particularly true of DIALOG which is loaded into a computer core many times in a typical simulation. Further, any looping or repetitive execution of user programs would result in the saving of computer time if stored in absolute element form on SIMTAP. This is true for repetitive executions in different jobs or the same job on different days.

Figure 5 is an illustration of how one sets up a run to create or update a SIMTAP. Units A and B are assigned as the new and old SIMTAP's. One tape could be used but it is usually advisable to use two in order to avoid losing previously stored programs in case of error. Unit C is a PCF tape containing the program or programs which are to be added to the new SIMTAP.

The first step in adding a new program to SIMTAP is to read the program tape for the desired program into the PCF using CUR. Those subroutines which need modification or have no relocatable binary elements must be compiled (recompiled).

TABLE I      ODIN SIMULATION AND DATA  
TAPES AT JSC

<u>TAPE</u>	<u>CONTENTS</u>	<u>DATA</u>	<u>SOURCE</u>	<u>CODE</u>	<u>ABS</u>
A11772	ODBIN				X
A05442	ODLOG				X
(SAVE)	OASOP				X
	OMAING				X
	OWAATS				X
	OWETED				X
	OPCFNT				X
	ODIN/XQT		X		
	INPUT/DATA		X		
	ODIN/SAVE		X		
A05989	AESOP/RBROC	X			
	LOOP TEST	X			
(SAVE)	ODIN WING	X			

7	ASG A = _____	New Simulation Tape
8		
7	ASG B = _____	Old Simulation Tape
8		
7	ASG C = _____	Program to be Added or
8		Updated (NEWPRGM)
7	ASG D = D	Scratch Unit
8		
7	XQT CUR	
8	TRW C	
	IN C	
	TRI C	
	( PROGRAM MODIFICATIONS )	
7	ABS NEWPRGM, NEWPRGM	(Create Absolute Element)
8		
7	XQT CUR	
8	TRW D	
	OUT D, 2	
	TEF D	
	ERS	
	TRW B	
	IN B	
	TRI B	
	TRW D	
	IN D	
	TRI D	
	TRW A	
	OUT A	
	TEF A	
	OUT A	
	TEF A	
	TRI A	
	TOC	

	⊙ Temporarily store absolute element for new or modified program.
	⊙ Load old simulation tape. Not essential for new tape.
	⊙ Load new or modified program tape.
	⊙ Create new simulation tape (2 files). This function may be performed by the control card below if the element exists.

7	ADD ODIN/SAVE
8	

FIGURE 5      ADD OR UPDATE AN ODIN PROGRAM

Next an absolute element must be created using the control card:

7<sub>8</sub> ABS NEWPGM,ABSPGM

NEWPGM is the name of the main program or loadable segment (for an overlayed program). ABSPGM is the absolute element program which will ultimately be added to SIMTAP. Any number of absolute elements programs may be created (or recreated) in the PCF. Once all the desired absolute element programs are created (or recreated), they are temporarily stored on Unit D, a scratch unit assigned for this purpose. A CUR operation is performed for this purpose.

OUT D, 2

places only the absolute element programs from PCF onto the scratch unit. Next the PCF is erased and the contents of the old SIMTAP (Unit B) are loaded into the PCF. Then the new programs stored on D are loaded into the PCF. Any programs already stored (in PCF) will, of course, be replaced. Now the PCF contains all the programs including the new or replaced ones in absolute element form. The next CUR operations are to place the contents of the PCF onto the new SIMTAP. One SIMTAP created at JSC for test and demonstration is shown in Table II.

#### Modification of the Control Card Data Base

The addition of new programs to the SIMTAP requires a modification to the control card data base (CCDATA) before the program can be used in a simulation. Most computer installations where ODIN is used utilizes a stored file called CCDATA which is retrieved each time the DIALOG executive system is used. At JSC, the EXEC II system on the Univac 1108 does not have a convenient means of storing CCDATA. For the EXEC II system, CCDATA will be created each time a simulation is initiated as illustrated in figure 4. Therefore, the modification of CCDATA will be performed on the data tape (figure 4, cards 8 through 20). The above data set describes the execution procedure for each program available on SIMTAP. When a new program is added to SIMTAP, a new execution procedure must be added. The "procedure" may be a single control card or several control cards:

MYPGM =

7<sub>8</sub> XQT ABSPGM

TABLE II UNIVAC 1108 (EXLC II) UNITS COMPATIBILITY CHART

		LOGICAL UNITS																												
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z			
PROGRAM		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
DIALOG (pass 1)																				1	1	1	1				S	1	1	
DIALOG														1						1	1	1	1				S	1	1	
NETTED								S				S		1																
MAATS														0																
COMPUT								2						0																
MALNG				0																										
RESOP										2				0																
PCFENT								S																						

KEY 0 = READ IN  
 1 = SOFT CODED  
 2 = HARD CODED  
 S = SCRATCH

The actual control card must start in column 7 (shifted by one computer word). The shifted control card or cards must be preceded by a user selected name unique to CCDATA by which the stored procedure will be referred during an ODIN simulation. For example:

&EXECUTE MYPGM&

The new procedure may be placed between any of the existing procedures. For example, it may be placed after card 10, 12 or 17 in figure 4. The order does not matter since they are recalled by name.

### Units Compatibility

The use of ODIN at JSC on the Univac 1108 (EXEC II) system requires that any unit which has data stored on it and which will be used by another program or the same program upon reexecution must be "reserved." That unit number can be used by no other program in the simulation. Units compatibility is not required on the EXEC VIII system since logical units can be simply reassigned. Users of EXEC II systems must assure units compatibility among programs in the simulation. This can be done conveniently by a table similar to the one shown in Table II. Here a list of several programs is tabulated in the left column and the units each uses is identified along the top. Several codes are used depending upon how the unit is coded and used.

Hard coded into the program such that major modification would be required to change the unit number which it uses.

Soft coded into the program such that the unit number is a variable name which is set in a data statement and could be changed with a simple modification.

Read into the program as input data such that no program modification would be required to change the unit number.

Scratch units such that the unit is used in the programs only as temporary storage.

A careful study of a units compatibility chart such as the one shown in Table II will be required to determine the modification, if any, which will be required to use a program in ODIN.

## CATALOGUING PROCEDURE

The DIALOG executive system uses many program procedures in the course of producing a simulation. Among these are the CCDATA file, the MODIN file and the COPY5 file. These files are catalogued procedures which are employed as required by the DIALOG executive to "create" a simulation based on user instructions. Similarly, the user of the DIALOG executive system has need for catalogued procedures when repetitive control card sequences are used over and over. The PCFENT program provides a means by which the user can enter procedures (including data) into the PCF to be used later via the ADD control card. Examples of this procedure are:

$$\begin{array}{l} \text{ODIN/XQT} = \left\{ \begin{array}{l} 7_8 \text{ ASG L,S,T,U,V,Y,Z} \\ 7_8 \text{ HDG - - - - -} \\ 7_8 \text{ XQT ODBIN} \\ 7_8 \text{ XQT ODLOG} \end{array} \right. \\ 7_8 \text{ ADD MODIN/1} \end{array}$$

used in the initialization of the DIALOG executive system:

$$\text{INPUT/DATA} = \left\{ 7_8 \text{ XQT MAING} \right.$$

used to modify input data on a data tape, and

$$\text{ODIN/SAVE} = \left\{ \begin{array}{l} \text{TRW A} \\ \text{OUT A} \\ \text{TEF A} \\ \text{OUT A} \\ \text{TEF A} \\ \text{TRI A} \end{array} \right.$$

used to write a PCF tape of ODIN programs and files.

## Use of PCFENT

Elements are entered into the PCF through the PCFENT program as follows:

7<sub>8</sub> XQT PCFENT

NAME/VERSION	(free field)
--------------	--------------

7 <sub>8</sub> XQT NAME1	(control cards starting in column 7)
--------------------------	--------------------------------------

( Other non-control cards start in column where used.
---

Control cards start in column 7. They are shifted to column 1 by the PCFENT before being stored in the PCF. PCFENT uses unit 8 for a scratch unit.

## Use of Catalogued Procedures

Once stored in the PCF, catalogued procedures may be added to the input stream at any time as follows:

7<sub>8</sub> ADD NAME/VERSION

The effect of the illustrated control card is to process the contents of "NAME/VERSION" before returning to the input stream for more data.